新一代变电站集中监控系统系列规范
第 4 部分：基础平台
（2023 年预发）

目　　次

前　言

为规范新一代变电站集中监控系统的功能和技术要求，为变电站集中监控系统的设计和研发提供指导和参考，特制定本规范。

本文件是《新一代变电站集中监控系统系列规范》的第4部分。《新一代变电站集中监控系统系列规范》发布以下部分：

——第1部分：总则；

——第2部分：设计规范；

——第3部分：数据规范；

——第4部分：基础平台；

——第5部分：功能应用 ；

——第6部分：人机界面 ；

——第7部分：检测规范；

——第8部分：远程智能巡视集中监控应用；

——第9部分：电网资源业务中台交互；

——第10部分：模型规范。

本规范起草单位：。

本规范主要起草人：。

新一代变电站集中监控系统
第 4 部分：基础平台

1 范围

本文件规定了新一代变电站集中监控系统（以下简称：集控系统）基础平台架构，规定了平台基础和支撑公共组件的功能、性能等技术要求。

本文件适用于 35kV-500kV 变电站接入的变电站集中监控系统基础平台建设，并指导系统的研发、验收和应用，可供其他电压等级变电站和集控系统建设参考。

2 规范性引用文件

下列文件中的内容通过文中的规范性引用而构成本文件必不可少的条款。其中，注日期的引用文件，仅该日期对应的版本适用于本文件；不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

GB/T 30149-2019 电网通用模型描述规范

GB/T 33601 电网设备通用数据模型命名规范

GB/T 36572 电力监控系统网络安全防护导则

DL/T 476 电力系统实时数据通信应用层协议

DL/T 634.5101 远动设备及系统 第 5-101 部分：传输规约基本远动任务配套标准

DL/T 634.5104 远动设备及系统 第 5-104 部分：传输规约-采用标准传输协议集的 IEC60870-5-101 网络访问

DL/T 860 变电站通信网络和系统

DL/T 890 能量管理系统应用程序接口

DL/T 1230-2013 电力系统图形描述规范

Q/GDW 1680.31-2014 智能电网调度控制系统 第 3-1 部分：基础平台 消息总线和服务总线

Q/GDW XXXXX.4—2022 高可靠变电站二次系统 站控系统技术规范 第 4 部分：数据通信网关机

Q/GDW XXXXX.7—2022 高可靠变电站二次系统 通用技术规范 第 7 部分：电气操作防误

新一代变电站集中监控系统系列规范 第 1 部分：总则（2023 年试行）

新一代变电站集中监控系统系列规范 第 3 部分：数据规范（2023 年试行）

新一代变电站集中监控系统系列规范 第 5 部分：功能应用（2023 年试行）

新一代变电站集中监控系统系列规范 第 6 部分：人机界面（2023 年试行）

新一代变电站集中监控系统系列规范 第 9 部分：电网资源业务中台交互（2023 年试行）

3 术语和定义

下列术语和定义适应于本文件。

3.1

广域范围 Wide Area

物理网络上两个区域不能通过直接连通的方式来实现业务数据交互，包括基于调度数据网实现交

互系统内延伸范围。

3.2

态 context

在一个与时间相关的运行环境下，达成某些确定目标的一组应用的集合。根据运行环境的不同可分为实时态、预演态、测试态、反演态。一个应用可以在不同的态中被复用。

3.3

应用 application

由一组互相紧密关联的功能组成，用于完成某一方面的业务工作。

4 平台功能要求

4.1 总体要求

系统基础平台应为各类应用的开发、运行和管理提供通用的技术支撑，提供统一的数据服务、模型管理、数据管理、图形管理，满足系统各项实时、准实时和生产管理业务的需求。基础平台应遵循DL/T 890标准，且满足以下技术要求：

a) 应提供对整个系统中应用的统一管理和协同工作，包括：系统节点及应用管理、进程管理、系统管理交互界面、时钟管理等；
b) 应具有良好的开放性、扩展性，提供标准的API接口和服务化接口，支持第三方应用的集成，消息总线和服务总线接口应符合Q/GDW 1680.31要求，API接口定义参见附录A，服务化接口定义参见附录B；
c) 为应用提供公共模型（见附录A的2.3 数据字典）与运行数据，负责模型和数据的跨安全区、跨层级传输与同步；
d) 为应用提供多样化的数据存储， 包括实时数据库、 历史数据库、 日志数据存储等；
e) 为应用提供全面的数据通信手段，包括服务总线、 消息总线和业务流程等，可通过平台实现横向、纵向的应用数据传输与共享；
f) 为应用提供统一的人机交互界面，具备嵌入应用定制界面的功能；
g) 具备转发数据和原始报文功能。

4.2 软件架构

集控系统软件架构由基础平台、功能应用和系统安全体系三部分组成。其中基础平台包括基础平台公共组件、基础平台支撑应用两部分。基础平台公共组件包括实时数据库、关系数据库、系统管理、模型管理、权限管理、告警服务、公共服务、跨区协同、系统备份与恢复、人机服务。基础平台支撑应用包括人机工具、图模维护工具和系统维护工具。系统安全体系参见《新一代变电站集中监控系统 第1部分 总则》，应遵循GB/T 36572的要求。具体基础平台架构见图1。

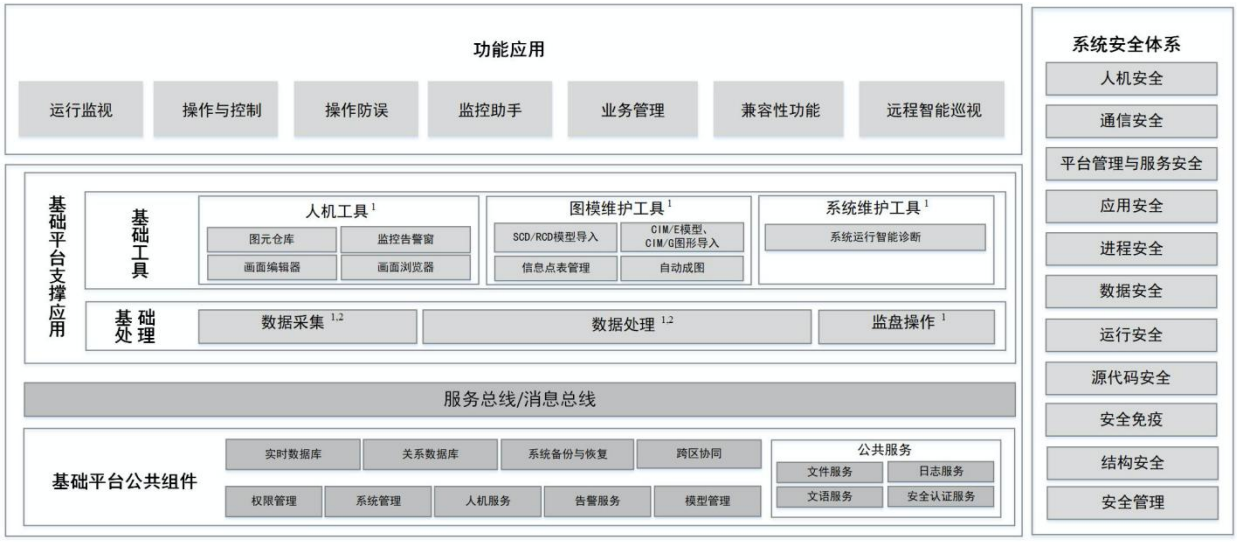


图 1　集控系统基础平台架构图

4.3　系统管理

4.3.1　系统节点及应用管理

4.3.1.1　功能要求

系统节点及应用管理应具备但不限于以下功能：

a)　系统任何单一故障，如单一节点、单一网络、单一应用故障，都不会导致系统主要功能的丧失或使系统性能低于要求的水平；

b)　当应用主机出现故障时，能自动将主机切换到正常的备用服务器。若正常的备用服务器存在两台或两台以上，应用状态在各服务器间保持同步，主机故障后备机中能够自动产生新的主机；

c)　可查询应用工作状态，可通过系统管理交互界面工具监视节点及应用的状态，并可进行主备切换及启动停止操作；

d)　应提供节点及应用运行风险告警功能，可以进行节点机磁盘、CPU、内存等异常风险告警；

e)　应提供节点及应用运行数据汇总与存储功能；

f)　宜支持冷备功能。当某个应用的在线运行节点数量少于 2 个时，可以自动在预先配置的节点启动应用形成热备用，否则自动停止该应用，处于冷备用状态。

4.3.1.2　接口要求

系统节点及应用管理具备应用运行状态信息获取接口。

4.3.2　进程管理

4.3.2.1　功能要求

进程管理应具备但不限于以下功能：

a)　应提供系统管理交互界面工具，可对服务器、工作站上运行的每个进程（平台进程、应用进程）进行监视和管理，能详细列出进程信息；

b)　当系统进程发生异常退出时，系统能够对其进行自恢复，并发出报警通知；

c)　应提供进程运行风险告警功能，可以进行进程 CPU、内存等异常风险告警；

d)　应提供进程运行数据汇总与存储功能。

4.3.2.2　接口要求

进程管理具备但不限于以下接口：

a) 进程注册、注销接口；
b) 获取进程运行信息接口；
c) 进程运行状态接口。

4.3.3 系统管理交互界面

系统管理交互界面为系统管理功能的统一入口，是对系统中节点、应用、进程等进行配置的工具，提供配置节点属性、应用运行方式、进程相关参数等功能，同时提供应用、进程、CPU、内存等状态的监视功能，并支持对应用、进程等进行操作，具体包括但不限于以下功能：

a) 应提供节点配置功能，可配置系统各节点，包括节点名称、节点类型等属性；
b) 应提供应用配置功能，可配置应用名称、功能以及应用的分布等属性；
c) 应提供进程配置功能，可配置进程名称、启动方式、进程级别等属性，进程按重要级别支持配置为关键进程、普通进程等；
d) 应提供数据库存储表空间等信息监视功能；
e) 应提供主备节点配置、宜具备冷备节点配置功能；
f) 应提供系统节点、应用、进程等的运行信息监视功能，可监视内存使用量、内存占用率、CPU 占用率、磁盘使用量、磁盘占用率等运行数据信息；
g) 应提供系统节点网卡名、网卡地址、网络流量、网卡状态等网络数据信息的监视功能；
h) 应具备查看各节点、节点应用、应用进程等运行状态的功能；
i) 应具备启停节点应用、节点进程操作功能，支持应用、进程的启动/停止等操作。

4.3.4 时钟管理

时钟管理应具备以下功能：

a) 系统能够接收时间同步装置的标准时间，时间同步装置能够选择北斗、GPS 作为时钟源，系统的各个工作站通过 NTP 协议和标准时间自动同步，保持系统的各个硬件设备的时间一致，同时能够避免闰秒的影响；
b) 对接收的时间信号的正确性应具有安全保护措施。当时钟源时间与本地时间偏差超过 3600 秒后，系统暂停对时，并产生告警。

4.3.5 多态多应用管理

多态应用管理应具备以下功能:

a) 支持多态、多应用管理能力，一个态可以包含多个应用，应用下包含若干进程；
b) 系统中支持的态不少于 4 个，如实时态、预演态、测试态、反演态；
c) 一个节点上可以运行多个不同的态；
d) 每个不同的态下可以单独配置不同的应用；
e) 可以配置进程在不同的态下运行；
f) 提供自定义扩展应用的配置功能。

4.4 关系数据管理

4.4.1 基本要求

关系数据是指存放在第三方商用数据库中的数据，主要用来保存变电站设备、参数、静态拓扑连接、系统配置、告警和事件记录、历史统计信息等。关系数据管理的历史数据包括但不限于：

a) 事件记录及各级各类告警信息；
b) 统计数据；
c) 设备运行数据；
d) SOE 数据；
e) 带时标数据。

4.4.2 历史数据管理

4.4.2.1 功能要求

历史数据管理具有以下功能：

a) 提供按时间对历史数据的导入、导出的功能；
b) 提供根据数据库存储空间的裕度发出告警的功能；
c) 提供基于时间范围的数据筛选功能。

4.4.2.2 界面要求

提供商用关系数据库的导入导出界面工具。

4.4.2.3 接口要求

具体见附录历史数据服务接口。

4.4.3 历史数据存储

4.4.3.1 功能要求

历史数据存储具有以下功能：

a) 应支持秒级和分钟级周期的遥测采样，周期包括：1 秒、5 秒、1 分钟、5 分钟、10 分钟、15 分钟、30 分钟；
b) 提供四遥、数据库操作、挂牌信息等事件记录存储功能；
c) 提供历史数据统计功能，包括最大/最小值及发生时间、平均值、负荷率等数据；
d) 统计类数据可按年、月、日等时段以及最大、最小等方式进行查询；
e) 提供商用库异常时历史数据缓存功能，提供商用库恢复后历史数据补存功能以及重新统计功能。

4.4.3.2 接口要求

API 接口具体见附录历史数据服务接口、商用库服务接口。

4.4.4 历史数据查询

4.4.4.1 功能要求

平台提供通用历史数据查询功能，人机客户端类应用禁止直接访问商用关系数据库；具体包括但不限于：遥测、遥信、告警事件等原始数据与统计分析数据的查询。

4.4.4.2 界面要求

历史数据查询界面应具备以下功能:

a) 具备在二维坐标系中展示数据变化曲线的功能；
b) 具备多条曲线能根据画面设置使用不同颜色展示曲线的功能；
c) 具备展示曲线图例，并且点击图例时可对曲线进行显隐的功能；
d) 具备曲线坐标系 Y 轴能根据曲线数据自动变化 Y 轴范围，且 Y 轴刻度在自动变化时自动规整的功能；
e) 具备按照画面设置展示不同采样周期曲线的功能。

4.4.4.3 接口要求

API 接口具体见附录历史数据服务接口、商用库服务接口。。

4.5 模型管理

4.5.1 功能要求

模型管理为系统提供全流程的模型管理服务，包括模型校验、模型变化通知、模型维护、模型同步等内容，应具备以下功能：

a) 模型维护功能：支持模型的增删改查操作。包括一次设备、二次设备、辅设备模型等数据的增删改查操作，插入操作时进行统一编码。唯一编码：规范各类型设备、各类型量测编码规则，各功能按照规则通过编码获取设备类型、量测类型；
b) 新增、修改、删除模型后通过消息总线发送模型修改消息，消息包含操作类型和变化模型 ID，关注的功能可订阅获取模型修改消息；
c) 模型数据同步数据库、实时库，应保证实时库与数据库的一致性；
d) 模型跨区同步，满足 IV 区模型访问需求；
e) 模型表的结构维护功能，支持一次设备数据结构、二次设备数据结构及辅助设备数据结构查询；
f) 模型表结构的导入和导出功能；
g) 模型数据管理通过设立通用数据校验规则，包括唯一性检验、约束关系、引用等，保证进入数据库的数据都是有效的，任何无效的数据将不允许进入数据库。

4.5.2 界面要求

模型管理需要提供模型浏览、表结构修改、数据导入导出界面工具，应具备以下功能：

a) 查看各应用，表中的数据；
b) 过滤数据，展示指定数据；
c) 修改、删除数据；
d) 导入、导出模型数据。

4.5.3 接口要求

模型修改接口见附录实时数据服务接口、模型修改服务接口：

a) 模型插入接口；
b) 模型修改接口；
c) 模型查询接口；
d) 模型删除接口。

4.6 安全管理

4.6.1 用户管理

4.6.1.1 功能要求

用户管理应具备以下功能:

a) 应具备用户的新增、修改、删除功能；
b) 应支持密码、智能卡或生物识别（指纹识别）等多种手段在内的用户认证；
c) 应支持设置用户密码策略，可对用户密码的长度、复杂度、有效期进行设置；
d) 应具备登录失败一定次数后锁定用户的功能，登录失败次数、锁定时间可设置；
e) 应具备给用户授予角色功能；
f) 应支持用户与权限的绑定，不同用户应按照工作范围、职责分工分配相应的访问控制权限；
g) 应提供权限添加、权限修改、权限删除等功能；
h) 应具备对用户进行唯一标识，且提供独立的身份验证模块，能够对所有用户进行多重身份鉴别；
i) 应具备用户密码规范性校验功能；
j) 应支持对用户登录时间信息存储。

4.6.1.2 界面要求

用户管理界面应具备以下功能：

a) 提供统一的用户管理操作界面，包含用户的增加、删除、修改、查询功能；
b) 提供系统安全管理策略配置界面，包含密码策略、登录失败策略等。

4.6.1.3 接口要求

提供用户登录身份鉴别接口，见附录权限服务接口。

4.6.2 角色管理

4.6.2.1 功能要求

角色管理应具备以下功能要求：

a) 应具备角色的新增、修改、删除功能；
b) 应支持角色与权限的绑定，不同角色人员应按照工作范围、职责分工分配相应的访问控制权限；
c) 应提供基于角色的权限添加、权限删除等功能；
d) 应提供系统管理员、审计员、业务操作员三种基本角色；
e) 系统管理员角色应仅具有用户管理、角色管理、权限管理等系统管理权限；
f) 审计员角色应仅具有审计数据的管理、监视的权限；
g) 业务操作员角色为系统的最终业务用户，不具有任何管理权限。

4.6.2.2 界面要求

提供统一的角色管理界面，满足以下要求：

a) 包含角色的新增、修改、删除、查询功能；
b) 包含角色与权限的绑定配置；
c) 包含角色与用户的绑定；
d) 包含修改角色权限可继承给角色下的用户。

4.6.3 权限管理

4.6.3.1 功能要求

权限验证和控制应具备以下功能：

a) 将用户需要的对象授权给用户，实现基于对象的验证和控制；
b) 通过用户、用户组和物理位置的关联，实现基于物理位置的权限控制；
c) 通过角色继承的方法，实现基于角色的权限控制；
d) 应支持从表域、表、数据库等多种数据粒度的权限验证；
e) 应具备对远方操作权限的校验功能；
f) 应保证系统管理员、审计员、操作员角色之间权限互斥，系统中不得存在超级管理员角色；
g) 应具备系统软件授权策略满足权限最小化原则，且满足权限互斥原则；
h) 应具备访问控制能力，依据安全策略控制主体对客体的访问。

4.6.3.2 界面要求

权限管理界面应具备以下功能：

a) 提供统一的登录及权限验证窗口界面，登录界面包含用户名、密码、双因子验证、登录时长设置，登录时长支持多个可选择项；
b) 提供统一的权限管理界面，满足权限管理功能要求。

4.6.3.3 接口要求

权限管理接口应具备以下功能：

a) 提供画面权限验证接口；
b) 提供数据库表权限验证接口；

c) 提供给定功能权限验证接口

4.6.4 审计管理

4.6.4.1 功能要求

审计管理应具备以下功能:

a) 应具备审计查阅、审计分析、审计报表等功能;
b) 应保证无法删除、修改或覆盖审计记录。

4.6.4.2 界面要求

提供审计日志调阅工具，支持对审计记录的搜索、查询、分类、排序等调阅审计操作。

4.6.4.3 接口要求

提供操作审计记录产生接口，如远方控制、关键模型的增删改、用户登录注销等应用操作可直接调用接口生成日志并可在界面查询。

4.7 实时数据库

4.7.1 功能要求

实时数据库应具有以下功能：

a) 应支持主辅设备模型按表等方式创建，保证主辅设备模型初始化与数据库一致;
b) 应支持主辅设备模型维护时，保证主辅设备模型变化与数据库一致;
c) 提供图形化的实时数据维护界面，对主辅设备模型和实时数据在线浏览、编辑;
d) 提供支持多应用的本地和网络访问接口进行实时库表记录的增删改查操作;
e) 提供主备机实时库间的数据同步功能，保证数据一致性;
f) 支持按照应用部署情况和配置策略，自动下装某应用对应节点的实时库数据;
g) 应具备主备机实时库负载均衡，在主备实时库间能够进行负载均衡;
h) 支持数据分片功能，数据库表可按 ID 分段等方式拆分成多个分片存储;
i) 支持主辅设备模型维护时，实时库数据变化通知功能;
j) 支持断面备份、恢复。

支持的基本数据类型应包括但不限于长整型、整型、短整型、浮点型、长浮点型、逻辑型、无符号字符、字符串型、时间型。

4.7.2 界面要求

提供图形化的实时数据维护界面，具备以下功能：

a) 对主辅设备模型和实时数据在线浏览与编辑;
b) 支持多种过滤方式查看数据;
c) 符合安全规范要求，支持用户登录及权限验证，具备审计功能。

4.7.3 接口要求

实时库提供多种接口，具体包括：

a) 具备快速存储和访问能力，提供高速的本地访问接口、远方服务访问接口;
b) 提供 C++，JAVA 等主流语言获取数据接口;
c) 提供读取、更新、删除、插入的实时数据库访问接口，支持读取全表、读取指定字段、按关键字查询、按属性值查询、按复合属性集查询、模糊查询等。

4.8 消息总线

4.8.1 功能要求

消息总线提供进程间（计算机间和内部）的信息传输支持，用于支持遥测、遥信等各类实时数据和事件的快速传递。消息总线应具有以下功能：

a) 具有注册/撤销、发送、接收、订阅、发布等功能，支持消息总线上一对一、一对多的消息传递；
b) 跨主机之间和主机内部进程间的消息传递；
c) 应保证消息的唯一性；
d) 应支持多个应用订阅相同的消息，应用间相互独立；
e) 提供消息总线监视功能，包括根据设定时间周期对收发消息数的统计，对消息总线数据阻塞等异常状态的监视，对已注册进程的正常运行、故障、退出等状态的监视，对事件集的订阅信息的查询等。

4.8.2 接口要求

提供包括但不限于以下类型接口：

a) 消息初始化；
b) 消息订阅；
c) 取消订阅；
d) 消息接收；
e) 消息发送；
f) 消息退出。

4.9 服务总线

4.9.1 基本功能

4.9.1.1 功能要求

服务总线是面向服务系统架构的基础，提供服务的接入、访问、查询等功能，实现了服务的灵活部署和即插即用。服务总线屏蔽网络传输、链路管理等细节，提供标准、开放的开发和集成环境，满足系统可扩展性、伸缩性的需求。服务总线应具有以下功能：

a) 定义面向服务的应用程序开发框架；
b) 提供服务的注册、定位、查询等功能；
c) 具备请求服务访问功能，支持应用请求/响应模式的局域/广域服务访问的功能；
d) 具备订阅服务访问功能，支持应用订阅/发布模式的局域/广域服务访问的功能；
e) 提供服务连接数、服务所在节点和服务端口等监视功能。

4.9.1.2 接口要求

提供包括但不限于以下类型接口：

a) 服务初始化；
b) 注册服务；
c) 服务定位；
d) 请求服务；
e) 响应服务；
f) 订阅服务；
g) 发布服务。

4.9.2 服务代理

服务代理实现广域范围的服务访问，服务代理应符合如下要求：

a) 配置形成广域范围的区域信息；
b) 代理之间应能实现负载均衡；
c) 实现本地和远程请求/响应、订阅/发布信息的交互；
d) 提供服务代理处理广域服务访问网络流量、频次和服务调用记录等监视功能。

4.10 公共服务

4.10.1 文件服务

4.10.1.1 功能要求

文件服务应提供以下文件管理功能和文件同步功能：

a) 支持创建、修改、查询、删除画面文件、点表文件等；
b) 支持画面文件、点表文件等文件版本功能：提供提交版本、按版本访问功能；
c) 支持画面文件、点表文件等文件列表查询功能；
d) 支持画面文件、点表文件等文件加锁功能，支持画面文件、点表文件等修改时的互斥操作；
e) 支持一主多备的文件安全存储和透明访问；
f) 支持目录的创建、拷贝、查询、删除等目录管理功能；
g) 具备安全 I 区到安全 IV 区的画面文件等正向文件同步功能；
h) 具备画面文件等广域传输功能；
i) 具备目录级文件过滤同步功能。

4.10.1.2 接口要求

应提供 C++的标准 API 接口：

a) 文件管理接口：创建、修改、查询、删除；版本管理；存储；本地访问、远程访问；
b) 支持单一进程内部对接口的并发调用。

4.10.2 日志服务

4.10.2.1 功能要求

日志服务应对各种应用产生的日志信息进行统一管理，应具有日志写入功能，可根据配置要求确定日志信息的处理方式。日志服务应提供以下功能：

a) 生成的日志文件存储在统一目录，且文件名由日志服务自动生成，同时支持多个进程对日志文件的并发访问；
b) 按文件大小滚动存储日志文件，且文件转储的大小上限可配置；
c) 记录的日志消息具有统一格式，并以优先级区分日志的紧要程度。

4.10.2.2 接口要求

应提供标准接口：

a) 日志存储接口；
b) 支持单一进程内部对接口 API 的并发调用。

4.10.3 安全认证服务

4.10.3.1 功能要求

安全认证服务应具备以下功能：

a） 应支持安全认证功能，支持对不同厂家提供的功能模块和服务的认证；
b） 应支持数据加解密功能，支持国密 SM4 算法的加解密；
c） 应支持防重放及防篡改功能，支持国密 SM2 算法的签名验签；

d） 应支持基于消息总线框架的安全消息传输；
e） 应支持基于服务总线框架的安全服务调用；
f） 流程宜符合《新一代变电站集中监控系统系列规范 第1部分 总则》附录A。

4.10.3.2 接口要求

提供包括但不限于以下类型接口：
a） 配置初始化；
b） 安全认证；
c） 服务请求加密；
d） 服务请求解密；
e） 读取密钥；
f） 数据加密；
g） 数据解密。

4.10.4 文语服务（可选）

4.10.4.1 功能要求

应具备提供文字转成语音文件功能。

4.10.4.2 接口要求

应提供语音合成接口。

4.10.5 操作控制服务

4.10.5.1 功能要求

操作控制服务应提供以下功能：
a) 主设备遥控预置、执行、撤销、直控，设点，调档预置、执行、撤销；
b) 辅设备遥控预置、执行、撤销、直控，设点；
c) 控制校验：应能校验遥控闭锁、挂牌等信息，对设备在控、禁止遥控等指令进行操作校验；
d) 应支持同一时间、同一厂站只有一个主设备可执行遥控操作；
e) 下发文件到子站；
f) 从子站读取文件；
g) 读取子站某目录下的文件列表；
h) RPC模型调阅；
i) RPC数据调阅；
j) 下发远方操作定值修改、召唤命令；
k) 二次设备服务，包括下列功能：召唤状态量、召唤模拟量、召唤定值区号、召唤故障录波文件列表、召唤故障录波文件、召唤装置历史信息、召唤装置通信信息、召唤软压板、召唤定值；
l) 顺控操作下发到子站；
m) 应支持返回错误校验码以及错误原因。

4.10.5.2 接口要求

操作控制服务应具备以下要求:
a) 遥控、设点、调档，参数定值修改，二次设备服务，其他服务（包括文件下发召唤、调阅文件列表、RPC 模型和数据调阅、读数据集）应提供基于服务总线的安全服务；
b) 顺控操作应提供基于消息总线的服务接口；
c) 操作控制服务接口。

4.11 告警服务

4.11.1 功能要求

4.11.1.1 告警信息管理

告警服务应统一管理应用产生告警的存储以及输出配置规则，应符合但不限于以下要求：

a) 统一告警类型，支持自定义扩展告警分类；在系统默认配置的告警类型基础上（见附录 A 告警服务接口），支持自定义扩展告警类型，包括必要的告警主类型、子类型等；
b) 统一告警信息数据结构，支持自定义扩展告警信息数据结构；
c) 告警信息，按对电网和设备影响的轻重缓急程度分为：事故、异常、越限、变位和告知五类；
d) 具备每类告警可定义不同的告警状态（如断路器的分和合，保护信号的动作和复归）；
e) 支持描述告警不同的处理方式，包括但不限于是否告警、是否上告警窗、是否音响/语音报警、是否推画面、是否存历史告警库等；
f) 具备不同等级的告警可定义不同的告警处理方式；
g) 具备对于同一告警类型的不同告警状态和不同发生对象（遥测/遥信）可支持不同告警处理方式。

4.11.1.2 告警接收

告警服务告警接收应具备以下功能:

a) 告警服务应提供统一的告警接收处理，统一接收应用发送的各级别告警信息，汇集基础平台自身告警及上层业务应用的告警；
b) 告警接收应具备统一的告警信息发送接口，应用可调用接口向平台发送告警信息。告警信息应符合规范的内容格式，告警对象包括但不限于告警级别、告警内容、告警时间等。

4.11.1.3 告警处理

告警服务告警处理应具备以下功能:

a) 告警服务收到应用发送的告警信息时，应根据预设的告警策略进行处理，根据告警管理配置规则，完成告警信息的实时输出。告警实时输出的属性内容包括：应用发送告警原始属性、告警显示颜色、告警等级、告警分类、告警语音、告警推图等；
b) 告警服务应提供统一的告警处理功能，包括但不限于对告警的确认操作。

4.11.1.4 告警存储

告警服务告警存储应具备以下功能:

a) 告警服务应提供统一的告警存储功能，对各基本告警、自定义扩展告警（类型及结构）进行统一存储到历史库；
b) 告警历史存储的属性内容包括：应用告警发送告警原始属性、告警确认状态/时间/用户、告警复归状态/时间、告警定制分类等告警信息应分类存储到历史库中。

4.11.1.5 告警订阅

告警服务提供全量实时告警的订阅发布功能，为订阅的各界面工具及应用服务主动推送实时告警。

4.11.2 界面要求

提供告警定义配置界面，详见 5.1.5 告警定义工具。

4.11.3 接口要求

具体见附录告警服务接口。

4.12 人机服务

4.12.1 数据刷新服务

4.12.1.1 功能要求

数据刷新服务应提供以下功能：

a) 刷新画面保证实时数据的正确性和及时性；
b) 具备浏览画面的数据刷新功能，支持自定义刷新周期，并根据刷新周期定时返回数据；
c) 画面数据刷新服务在响应客户端的请求时，组织与画面相关的实时数据，将数据推送到画面上；
d) 画面动态刷新功能采用订阅/发布的服务模式，按画面刷新周期返回变化数据；
e) 画面服务具有并发处理功能，可同时响应多个用户调用画面的并发请求；
f) 有缓存画面实时数据集的功能，能同时响应多个人机工作站上同一画面动态数据的刷新请求，避免数据库的重复访问。

4.12.1.2 接口要求

数据刷新服务接口应具备以下功能:

a) 数据刷新服务的数据输入包括：需要访问的态、应用、表、域等参数；
b) 数据刷新服务的数据输出包括：访问是否成功标识，返回的数据类型、长度等信息，以及返回的数据值。

4.13 跨区协同

4.13.1 功能要求

应为系统内部Ⅰ区与Ⅳ区的运行监控与业务应用提供跨区协同的通用处理，包括但不限于：

a) 模型数据同步功能：I区维护的模型数据自动同步至Ⅳ区；
b) 实时数据同步功能：I区实时数据变化后实时同步至Ⅳ区；
c) 实时告警同步功能：I区实时告警数据实时同步至Ⅳ区，告警事件时标一致；
d) 文件双向同步功能：
   1) Ⅰ区向Ⅳ区可跨正向隔离发送指定文件；
   2) Ⅳ区向Ⅰ区可跨反向隔离发送符合标准格式的E文件。
e) 商用库数据表同步：Ⅰ区、Ⅳ区业务应用分析生成的历史数据存储在本区商用库中，Ⅰ区历史数据可跨区同步至Ⅳ区商用库；
f) 支持网页形式发布集控系统的厂站图、间隔图。

4.14 系统备份与恢复

4.14.1 文件备份与恢复

4.14.1.1 功能要求

系统备份/恢复管理功能应符合以下要求：

a） 工作站与服务器的文件备份/恢复提供可视化的界面工具；
b） 工作站与服务器的文件备份/恢复的对象可配置；
c） 工作站与服务器的文件备份/恢复提供增量与全量备份/恢复功能；
d） 备份、恢复任务执行结果有日志记录。

4.14.2 数据备份/恢复

4.14.2.1 功能要求

数据备份/恢复管理功能应符合以下要求：

a） 备份/恢复提供图形化管理和命令行工具；
b） 支持电网设备模型和历史数据备份；
c） 备份、恢复任务执行结果有日志记录；
d） 支持按模型表选择备份与恢复；
e） 对于历史数据，可提供基于天的备份与恢复功能。

4.15 安全区 IV 统一门户

4.15.1 单点登录

单点登录应具备以下功能：

a) 应具备统一的用户登录页面，登录页面包含用户名、密码、验证码，登录成功后自动跳转至该用户配置的应用首页；接入平台门户的应用页面应满足单点登录集成要求，在用户没有登录或登录超时情况下访问应用页面时，应用页面应先跳转到用户登录页面，登录成功后跳转回应用页面；

b) 应具备用户登录状态共享功能，在用户已登录情况下，实现门户框架页面与支持单点登录的应用页面之间、支持单点登录的不同应用页面之间免登录跳转；

c) 应具备用户退出状态共享功能，在门户框架页面发起用户退出后，已打开的支持单点登录的业务页面也将退出不能操作；

d) 应可使用安全区 I/II 的用户账号进行登录；

e) 接入平台门户的应用应基于平台 Token 校验服务，实现单点登录功能。

4.15.2 门户框架页面

门户框架页面应具备以下功能：

a) 应具备统一用户退出功能；

b) 应具备系统各业务模块菜单的分级导航和画面展示功能；

c) 应具备通过导航菜单打开业务页面并在展示区域展示。

4.15.3 访问权限控制

访问权限控制应具备以下功能：

a) 应具备安全区 I/II 用户账号同步至安全区 IV 功能，同步信息应包括但不限于用户登录名、用户姓名、所属用户组，安全区 I/II 用户账号密码可在安全区 IV 登录使用；

b) 用户管理员可在安全区 IV 独立创建用户账号功能，安全区 IV 创建的账号不可在安全区 I/II 登录使用；

c) 应具备业务页面、业务功能等业务资源的定义与管理功能 ；

d) 应具备基于角色的授权机制，实现用户对业务页面、业务功能等业务资源的访问权限配置。

4.15.4 接口要求

单点登录、权限访问控制接口应具备以下功能：

a) Token 校验；

b) 基于 Token 获取用户信息；

c) 基于 Token 获取用户菜单权限信息；

d) 基于 Token 判断用户是否具有某个功能的访问权限。

4.16 系统互备

基于两套同构平台进行系统互备建设，主、备系统独立采集相应区域数据。分别进行主辅设备监控与管理；通过模型数据、图形文件、系统配置参数、人工操作等数据同步实现主、备系统数据一致性，从而支撑系统业务双活。人机工具与系统、物理场所解绑，实现与主、备系统无差别访问。

4.16.1 功能要求

a) 应具备数据同步功能。同步类别包含但不限于模型、图形与人工操作。数据同步范围可配置，数据发生变化时，模型数据由主用系统同步到备用系统，图形文件和人工操作可进行双向同步，同步过程异常时发送告警；

b) 应具备数据一致性校验功能。数据校验以主用系统数据为基准。校验类别包含但不限于模型、图形与实时数据。数据一致性校验范围可配置，记录校验结果信息，当数据不一致时发送告警；

c) 具备以下主备管理功能：

1） 具备对两套系统的主备状态查询功能；
2） 具备对两套系统的主备状态手动切换功能；
3） 具备系统运行异常感知功能，并发送系统异常告警；
4） 主用系统允许下发控制类指令，备用系统禁止下发控制类指令。

4.16.2 界面要求

基础平台应提供技术支撑，在界面交互方面满足以下要求：
a) 具备选择访问主备系统功能；
b) 具备展示当前访问系统主备状态功能；
c) 具备展示各系统运行状态功能；
d) 具备数据一致性校验展示功能；
e) 具备数据同步展示功能；
f) 具备手动切换系统主备状态功能。

5 基础支撑要求

5.1 人机工具

5.1.1 基本要求

人机基本工具应提供基于 CIM/G 标准的统一的编辑、展示框架和开放的图形画面结构标准，开放集成框架下的人机界面开发，支持第三方应用界面集成，实现主辅关键信息一体化展示、控制和管理。分为图元仓库、画面编辑工具、画面浏览工具。

5.1.2 图元仓库

5.1.2.1 功能要求

5.1.2.1.1 基本图元

基本图元可直接用于绘制画面，包括：
a) 线：直线、弧线、折线、自由线等；
b) 基本形状：圆、椭圆、矩形、三角形、多边形等；
c) 文本；
d) 外部图片文件；
e) 动态数据。

5.1.2.1.2 电力系统设备图元

电力系统设备图元应符合《新一代变电站集中监控系统系列规范 第 6 部分 人机界面》要求。

5.1.2.1.3 综合类图元

综合类图元包括表格组件、曲线组件、棒图组件、饼图组件、仪表组件等，具体要求如下：
a) 表格组件为将数据库（包括实时数据库和关系数据库）中的数据以列表方式展示的图元组件，编辑时应支持关联实时库或关系库数据源、定义数据排序等功能，浏览时支持显示多种数据类型数据、记录查询、过滤、导出等功能；
b) 曲线组件应支持实时曲线、历史曲线，编辑时具备修改 X/Y 轴刻度参数、查询时间范围等功能，浏览时具备翻页（前一日、后一日）、查看曲线数据列表、修改曲线数据等功能；
c) 棒图组件应能通过圆柱长度来表达数值大小，可分为多段，适合表达关联数据的相对关系；
d) 饼图组件应能分为多份，显示部分对整体的比例，适合用来表达百分比类型的电力系统数据，饼图上应能显示百分比、每部分能配置不同的颜色；
e) 仪表组件应能通过指针式仪表显示当前值和不同区段的限值信息，通过指针角度反映数值变

化。

5.1.2.1.4 图元编辑

图元编辑器应具备以下功能:

a) 图元应基于矢量技术，支持平滑缩放；
b) 图元支持坐标、旋转、拉伸、镜像等属性设置；
c) 提供图元编辑功能，支持电力设备图元的创建、编辑等操作；
d) 支持多状态图元编辑。

5.1.2.2 界面要求

图元编辑器界面应具备以下功能:

a) 电力系统设备图元编辑器应具备图元的创建、修改、存储功能；
b) 图元编辑器应提供基于基本图元的编辑功能，包括放大、缩小、复制、粘贴、移动等操作，可为组件配置属性参数，支持对多状态图元的编辑和存储，存储格式应为 CIM/G 格式。

5.1.3 画面编辑器工具

5.1.3.1 功能要求

平台提供基本的画面编辑工具，实现基本的绘图框架，可绘制基础接线图、间隔图等图形，画面类型可自定义扩充。画面编辑工具应支持业务插件扩展，业务第三方应用可通过业务插件方式实现业务功能扩展；画面编辑工具应支持集成第三方应用程序，第三方应用程序可通过外部命令调用等方式，融入到系统中，编辑器保存的图形满足 CIM/G 规范。画面编辑器工具主要包括但不限于以下功能：

a) 文件操作：
   1) 支持创建、编辑、保存各类型画面文件；
   2) 支持画面文件依据集控站区域和图形类型保存在不同目录下；
   3) 支持所编辑画面文件客户端与服务器同步保存；
   4) 支持所编辑画面文件的锁定与解锁。
b) 布局显示：
   1) 支持画面放大、缩小、拖拽移动等缩放移动操作；
   2) 支持画面宽、高、背景等画面属性编辑与显示；
   3) 宜支持画面裁剪自动调整画布大小；
   4) 支持多个图元的上、下、左、右等对齐操作；
   5) 支持多个图元之间的间距调整操作。
c) 图元使用：
   1) 支持添加使用电力设备图元，可对其属性进行编辑，通过拖拽方式建立图模关联；
   2) 支持图元属性的显示与编辑，比如坐标、缩放、颜色等；
   3) 支持批量修改多个同类图元属性；
   4) 支持单个或者多个图元的移动，并且能够识别移动为平行或垂直移动；
   5) 支持图元鼠标悬浮窗口提示功能；
   6) 支持图元的批量选择功能，如按图元类型等；
   7) 支持图元间连接线校正功能；
   8) 支持图元的移动、旋转、拉伸、镜像、对齐等尺寸调整、置顶置底等操作；
   9) 电力系统设备图元应支持多状态显示，支持使用饼图、棒图、曲线、表格等常用综合图形；
   10) 支持多个图元组合，并对组合后的图元进行移动、复制等操作。
d) 图形管理：
   1) 画面维护网络保存后，应具备及时通知画面更新的能力。
e) 其他功能：
   1) 支持功能快捷键，如保存，打开文件，复制粘贴、撤销回退等；
   2) 支持本图或者跨图的单个或者多个图元的复制粘贴、带状态粘贴；

3） 支持编辑状态下画面着色功能；
4） 支持定制设备所附带量测，并且附带量测随设备移动，生成设备模型信息的同时自动维护画面量测信息；
5） 支持在画面中编辑母线、断路器、变压器等设备图元时，通过输入必填信息生成数据库模型记录；
6） 支持以列表方式显示图元必填信息并支持编辑；
7） 支持将常用电气元件的组合一起添加到画布，支持通过提前维护好这些元件之间的关联关系实现对电气元件组的统一维护；
8） 支持在画面中修改母线、断路器、变压器等模型记录；
9） 支持在画面中通过图形连接关系维护模型拓扑节点编号；
10) 支持对画面设备图模关联、连接点空挂等信息校验；
11) 支持通过格式刷快速记录、设置图元的部分公共属性；
12) 支持图形缩放比例与图层的联动，在不同的缩放比例时显示不同的图层；
13) 支持第三方应用插件集成。

5.1.3.2 界面要求

提供绘图界面，可绘制各类基础图形。

5.1.3.3 接口要求

第三方插件集成接口，详细见附录人机扩展接口。

5.1.4 画面浏览器工具

5.1.4.1 功能要求

平台提供通用的画面浏览工具（简称浏览器）应提供基本的图形展示、数据刷新功能，实现基本的人机展示框架，应具备以下功能:

a) 应具备良好的交互能力，应用可接收浏览器的各种操作事件；
b) 应提供右键菜单扩展功能，并支持业务触发扩展的右键菜单等条目时通知各业务，扩展接口，附录 人机扩展接口；
c) 应支持应用以业务插件方式扩展人机业务功能，浏览器加载应用插件，插件可嵌入到已有视图（图形画面中设定部分区域嵌入插件）；
d) 支持集成第三方应用程序,第三方应用程序可通过外部命令调用等方式，集成到系统中；
e) 浏览器工具还应包括但不限于以下功能：
1） 支持解析并显示 CIM/G 格式的图形文件功能；
2） 支持读取远端系统的图形及数据功能；
3） 支持画面动态数据点的自动刷新功能；
4） 支持网络拓扑着色功能；
5） 支持多种应用在不同态下的画面显示功能，并可任意切换；
6） 提供打开画面选择功能， 可以自由选择需要浏览的画面；
7） 支持锁定应用功能，切换画面时需要保持指定的应用状态；
8） 支持画面导出图片、打印功能；
9） 支持自由定义快捷键功能；
10) 提供循环调图功能；
11) 支持设备根据设备状态显示或隐藏功能；
12) 提供展示登录用户信息、登录节点信息、登录时间功能；
13) 支持画面框选缩放、滚轮缩放、全屏展示等功能；
14) 支持显示水印功能；
15) 支持展示曲线、表格、饼图、棒图等复合图元功能；
16) 支持画面文件快速检索打开画面功能；

17) 支持菜单、工具栏的自由定义与显示；
18) 支持右键点击图元时，弹出菜单功能；
19) 满足点击菜单选项可以执行相应的动作功能；
20) 支持鼠标悬浮在图元上方时，弹出设备信息提示信息；
21) 能同时显示多个窗口，支持窗口层叠、平铺，窗口大小可任意缩放；
22) 窗口显示区域可扩展到多个显示器，支持窗口跨显示器的拖动；
23) 支持窗口在多个桌面同时显示；在多显示器环境中，窗口可打开于指定的任一显示器上；
24) 画面网络保存成功后，浏览器应具备及时接收并提示画面更新的能力；
25) 支持应用打开指定画面、定位画面对象等。

5.1.4.2 界面要求

提供人机主菜单及工具栏的配置界面。

应符合《新一代变电站集中监控系统系列规范 第6部分 人机界面》界面要求。

5.1.4.3 接口要求

右键菜单扩展接口，参见附录人机扩展接口。

5.1.5 告警定义工具

5.1.5.1 功能要求

告警定义工具应提供以下功能：

a) 告警类型定义，可新增告警类型，并指定告警存储表；
b) 告警方式定义，可对指定告警类型、告警状态定义指定的告警行为（告警动作的集合）；
c) 告警行为定义，可对告警行为定义要执行的告警动作，支持的告警动作包括但不限于推画面、上告警窗、存历史库等。

5.1.5.2 界面要求

告警定义工具应具备以下功能:

a) 应提供界面，展示所有的告警类型、告警行为、告警动作信息；
b) 应提供新增/修改告警类型界面；
c) 应提供新增/修改告警行为界面；
d) 应提供新增/修改告警动作界面。

5.1.6 监控告警窗

5.1.6.1 功能要求

监控告警窗实时展示主辅设备告警信息并支持对告警进行相关操作，应具备以下功能:

a) 监控告警窗应包含但不限于以下基本功能：
   1) 告警信号展示遵循信号分类规范，分别显示“事故、异常、越限、变位、告知”等信号，显示内容可进行个性化配置；
   2) 支持单条告警确认、批量告警确认；
   3) 支持告警窗锁定与恢复功能；
   4) 支持将全部告警、未确认告警、未复归告警、已确认未复归告警的分类显示，支持告警确认后的颜色配置；
   5) 支持基于责任区的信息分流，告警窗上只展示登录用户责任区范围内的告警；
   6) 支持通过告警记录直接查看告警所属变电站主接线图或间隔图；
   7) 应具备自定义告警展示窗口功能，根据自定义条件从告警窗中抽取符合条件告警信息在自定义窗口中展示，原窗口内容不变；
   8) 应具备按条件及关键字过滤检索功能；
   9) 支持视图保存功能，根据监视需求自定义布局视图并保存，支持视图一键切换；

10) 告警窗信息应具备信息压缩功能，同一信息频发时只显示最新一次动作记录及动作次数；
11) 支持设置告警缓存数量或缓存周期功能。

b) 监控告警窗应包含但不限于以下自定义扩展功能：
宜支持告警事项操作菜单扩展功能，可以按需扩展菜单项。

5.1.6.2 界面要求

应符合《新一代变电站集中监控系统系列规范 第7部分 人机界面》的“6 集控站层监控界面”章节的“6.4 告警窗”界面要求。

5.1.6.3 接口要求

监控告警窗提供告警信息右键操作菜单扩展接口。

5.1.7 告警查询工具

5.1.7.1 功能要求

告警查询工具应具备以下功能：

a) 应具备多种历史记录过滤方式，包括但不限于责任区/运维班、变电站、电压等级、时间段、告警级别、确认状态、告警内容模糊查询等条件组合过滤；
b) 应具备将告警筛选条件组合保存成自定义告警模板的功能；
c) 应具备对告警记录进行多表综合查询和单表查询的功能；
d) 查询结果支持以CSV文件格式导出。

5.1.7.2 界面要求

告警查询工具界面应具备以下功能:

a) 查询时间选择；
b) 运维班、变电站、电压等级等过滤条件选择；
c) 查询结果展示。

5.2 图、模维护工具

5.2.1 SCD/RCD模型导入

5.2.1.1 功能要求

SCD/RCD模型管理功能要求如下：

a) 应具备解析SCD模型功能，并能够提取一次设备、二次设备、辅助设备模型以及测点信息；
b) 应具备一、二次设备以及拓扑关系导入功能，测点信息能与一、二次设备、辅助设备正确关联，并且能与RCD文件建立映射关系；
c) 应具备SCD/RCD增量比对导入功能，能够显示出增、删、改记录，并增量入库；增量导入修改时仅修改SCD/RCD涉及的属性，对应用自定义扩展的属性不做修改，不影响人工封锁、标志牌操作等监盘操作功能的属性；
d) 应具备对导入的站端原始SCD/RCD文件的版本管理功能，版本为集控内部文件管理版；
e) 应具备对导入的站端原始SCD/RCD文件查询、删除功能；
f) 应能提供其他应用获取模型文件的接口。

5.2.1.2 界面要求

SCD/RCD模型管理界面应该具备以下功能：

a) 选中指定文件，文件解析后内容展示；
b) 文件查询、删除；
c) 文件版本信息的查询、浏览；
d) 比较展示增、删、改信息。

5.2.1.3 接口要求

通过文件服务接口获取 SCD/RCD 模型文件。

5.2.2 CIM/E 模型、CIM/G 图形导入

5.2.2.1 功能要求

应提供模型导入工具，能够导入调度系统提供的存量站设备模型，并满足如下功能要求：

a) 支持一次设备模型、前置模型导入；
b) 支持电网拓扑关系模型导入；
c) 设备命名应符合 GB/T 33601《电网设备通用数据模型命名规范》，模型文件应符合 GB/T 30149-2019《电网通用模型描述规范》，并在 CIM/E 模型描述基础上扩充遥测、遥信、遥控信息；
d) 支持增量导入；
e) 应支持 CIM/G 图形文件导入功能，并能将图元关联到模型，图形文件应符合 DL/T 1230《电力系统图形描述规范》规范的相关要求。

5.2.2.2 界面要求

界面应具备以下功能:

a) 选择导入的模型文件；
b) 比较展示增、删、改信息。

5.2.3 信息点表管理

5.2.3.1 功能要求

信息点表管理应具备以下功能:

a) 变电站应提供符合《新一代变电站集中监控系统系列规范 第 3 部分 数据规范》的 RCD 文件，信息点名称应与变电站 SCD 文件中的装置模型测点名称保持一致；
b) 集控系统可根据变电站的 RCD 文件挑选生成集控系统信息点表；
c) 集控系统应支持信息点的信号取反等信号属性配置功能，并支持将信息点表及信息属性按需将全表或指定列的导入、导出功能；
d) 挑点后形成的 RCD 运行点表信息可下发给通信网关机，经由网关机现场确认后激活生效，文件交互方式参见 Q/GDW XXXXX.4—2022《高可靠变电站二次系统 站控系统技术规范 第 4 部分：数据通信网关机》附录 E；
e) 应提供集控系统与变电站信息点表的校核功能，通过与变电站 RCD 文件进行比对，实现信息点表的相互校核，并展示比对结果；
f) 与变电站交互点表应通过独立的管理通道实现。

5.2.3.2 界面要求

信息点表管理界面应具备以下功能:

a) 信息点表管理应提供友好的人机界面，提供对变电站 RCD 点表文件进行召唤、解析、校核、定制、保存、下发等功能；
b) 具备增量信息对比展示界面。

5.2.4 自动成图

自动成图宜满足如下功能要求：

a) 宜根据变电站一次设备模型及主接线模板自动生成主接线图：
   1) 保持自动生成的接线图风格一致；
   2) 根据设备模型和拓扑关系自动构建变电站内部设备连接关系，并识别出间隔组成；
   3) 能提供配置化方式设置变电站主接线图内间隔布局距离；
   4) 能根据变电站内部设备连接关系自动绘制变电站主接线图；

5) 自动生成的变电站主接线图应能自动绑定设备模型；
6) 自动生成的变电站主接线图应支持人工编辑。

b) 宜根据主接线图生成间隔分图：
1) 能根据变电站间隔典型接线方式，按需定制通用展示模板；
2) 能根据一次接线方式拓扑生成间隔实体图；
3) 自动生成的间隔分图应支持人工编辑；
4) 能根据模板中预定义的保护信息展示规则，实时获取、动态生成光字牌、压板图。

5.3 数据采集

5.3.1 数据采集功能要求

5.3.1.1 功能要求

数据采集功能应符合以下要求：

a) 支持对变电站一次设备、二次设备以及辅助设备等数据的采集和处理；
b) 支持下发对变电站的远方控制、调节和参数设置等命令，在正常数据召唤和传送时，如有控制命令需要传送，优先处理控制命令；支持选控、直控两种不同控制模式；
c) 支持二进制或 BCD 码模拟量的采集，支持系数及偏移量的处理；
d) 支持从Ⅰ区实时网关机接收数据、下发控制操作指令；
e) 支持从Ⅱ区服务网关机订阅数据、召唤文件、下发控制操作指令；
f) 支持 DL/T 634.5104、DL/T 476 通信报文协议，应支持客户端/服务器的角色；
g) 支持 DL/T 860 通信报文协议，应支持客户端角色；
h) 能通过多种/多个远动通道采集同一变电站的数据，支持进行通道优先级设置和数据处理；
i) 支持前置服务器多分组模式，支持运行系统在线动态扩容前置服务器和前置分组；
j) 支持集控站对厂站的时间同步监视功能，异常时发送系统告警；
k) 支持转发变电站上送的 DL/T 634.5101、DL/T 634.5104、DL/T 476 规约原始报文；
l) 支持通过 DL/T 634.5104、DL/T 476 规约以整厂或挑点方式转发实时数据；
m) 支持采集变电站网关机或网络安全装置上送的网络安全事件，并发送给应用
n) 应根据应用所需，为应用召唤变电站侧信息提供功能支撑，包括但不限于接收应用操作命令，完成变电站数据总召命令、顺控票召唤、录波文件召唤、画面调阅、设备运行状态调阅、历史数据调阅、一次/二次设备在线监测数据调阅等；
o) 应根据应用所需，在接收到变电站相应信息后，主动通知应用，包括但不限于接收顺控票、接收录波文件、接收 SCD/RCD 模型文件、接收二次设备版本信息等等；
p) 宜采用 Q/GDW XXXXX.7《高可靠变电站二次系统 通用技术规范 第 7 部分：电气操作防误》支持防误数据采集功能，包括但不限于防误接地线运行状态、网门运行状态、防误逻辑公式信息等。

5.3.1.2 接口要求

见附录操作控制服务接口、数据采集转发变电站网络安全事件接口。

5.3.2 数据采集管理

5.3.2.1 功能要求

数据管理功能应符合以下要求：

a) 监视通讯链路的运行情况，包括主/备通道的运行状态、误码率、停运时间、收发数据字节统计等；
b) 可自动统计各通讯链路的运行情况，包括通讯链路运行时间、运行率等信息；
c) 当数据通讯异常时应发出告警，包括通讯链路中断、数据传输中断、数据质量异常等情况；
d) 当告警产生后，根据应用要求、通信规约特点和用户配置，可采取手动或自动复位链路重新连接、切换采集主备机、切换链接等措施；

e) 支持选择当前值班通讯链路功能，支持根据通道优先级、通道运行状态等，自动切换值班通讯链路；支持手动切换值班通讯链路功能；
f) 可自动保存及人工定义条件保存通讯链路收发报文，对于控制类重要报文可单独自动保存；
g) 保存的报文应带接收报文时的时标；
h) 保存的报文应可实现滚动存储且周期可调。

5.3.2.2 界面要求

数据采集功能应提供丰富、友好的人机界面，供运行和维护人员对数据通信进行监视和控制，至少包括以下画面或界面：

a) 实时链路状态监视，如实时数据、通道链路状态等画面；
b) 应提供通讯链路管理操作界面，可在界面上对通讯链路实施启动、停止、主备通道切换操作；
c) 可维护通讯链路的各项配置参数，根据不同的通信规约应提供详细、完整、友好的配置界面；
d) 可在线监视报文，应提供友好的界面工具。

5.4 数据处理

5.4.1 模拟量处理

5.4.1.1 功能要求

对模拟量的处理应实现以下功能：

a) 应提供数据合理性检查和数据过滤；
b) 应能进行零漂处理，且模拟量的零漂参数可以设置；
c) 应能进行限值检查。每个测量值可具有多组限值对，用户可以自行定义限值对的等级，不同的限值对可以根据不同的时段进行定义，可以定义限值死区；
d) 应能进行数据不变化、数据跳变的检查并给出告警；
e) 应支持质量码处于人工封锁数据状态的数据处理，用人工输入值代替采集数据，展示并写入数据库；
f) 所有人工设置的模拟量应能自动列表显示，并能根据该模拟量所属变电站调出相应接线图；
g) 应将模拟量的处理结果通过消息总线的方式发送给应用。

5.4.1.2 接口要求

模拟量的处理结果通过消息总线的方式发送给应用。

5.4.2 状态量处理

5.4.2.1 功能要求

状态量的处理应完成以下功能：

a) 单点状态量用 1 位二进制数表示，1 表示合闸（动作/投入），0 表示分闸（复归/退出）；
b) 一次设备状态量应支持双位遥信处理，主、辅遥信变位的时延在一定范围（可定义）之内时状态正常，指定时延范围内只有一个变位则判定状态量可疑并告警，当另一个遥信上送之后可判定状态量由错误状态恢复正常；
c) 应支持质量码处于人工封锁数据状态的数据处理，用人工输入值代替采集数据展示并写入数据库；
d) 应将变化量的处理结果通过消息总线的方式发送应用。

5.4.2.2 接口要求

处理结果通过消息总线的方式发送给应用。

5.4.3 非实测数据处理

非实测数据可由人工输入也可由计算得到，以质量码标注，并与实测数据具备相同的数据处理功

能。

5.4.4 数据质量码

应对所有模拟量和状态量配置数据质量码，以反映数据的质量状况。图形界面应能根据数据质量码以相应的颜色显示数据。数据质量码至少应包括以下类别：

a) 未初始化数据：前置未上送过实时数据；
b) 计算数据：公式或其他计算结果数据；
c) 非实测数据：量测在前置未定义点号或通道号定义不完整；
d) 采集中断数据：前置与站端通讯中断；
e) 人工封锁数据：量测值被封锁，收到前置实时数据后不会更新；
f) 可疑数据：量测数据出现此状态表示遥测遥信不匹配，计算数据出现此状态表示公式计算中有分量的状态异常；
g) 控制闭锁数据：量测不允许控制操作；
h) 旁路代数据：量测数据被旁路替代；
i) 对端代数据：量测数据被对端替代；
j) 不刷新数据：在定义的时间周期内前置未收到量测报文的数据；
k) 越限数据：量测超过定义的限值；
l) 越合理值数据：数据超过合理值；
m) 告警抑制：禁止告警标记；
n) 三相不一致：三相开关位置状态不一致。

5.4.5 旁路代替

应提供以下旁路代替功能：

a) 旁路代替应能根据网络拓扑以旁路支路的量测值代替被代支路的量测值，作为该点的最终值进行显示，并以数据质量码标示旁路代替状态；
b) 提供自动和手动两种旁路代替方式；
c) 提供旁路代替结果一览表，可按区域、变电站、量测类型等条件分类显示。

5.4.6 对端代替

应提供以下对端代替功能：

a) 具备对端代替功能，当线路一端量测值无效时能以线路另一端的量测值代替，作为该点最终值进行显示，并以数据质量码标示对端代替状态；
b) 具备多端线路量测值汇总计算替代功能，如 T 接线；
c) 提供自动和手动两种对端代替方式；
d) 提供对端代替结果一览表，可按区域、变电站、量测类型等条件分类显示。

5.4.7 事件顺序记录

事件顺序记录（SOE）满足以下功能要求：

a) 以毫秒级精度记录一次设备状态、二次设备状态、辅助设备状态的动作顺序及动作时间，并形成事件顺序表；
b) 事件顺序记录应包括记录时间、动作时间、变电站名、事件内容和设备名；
c) 能根据类型、变电站、设备、动作时间和接收时间等条件对事件顺序记录分类检索和显示。

5.4.8 动态拓扑分析和着色

动态拓扑分析和着色满足以下功能要求：

a) 网络拓扑着色功能应能根据实时拓扑，确定系统中各种电气设备的带电、停电、接地等状态，并能够将结果在人机界面上用不同的颜色表示出来，包括：
   1） 不带电的元件统一用一种颜色表示；

2） 接地元件统一用一种颜色表示；
3） 正常带电的元件可根据其不同的电压等级分别用不同的颜色表示；
4） 当元件部分带电时，能正确进行拓扑着色。

b) 动态拓扑着色应能由事件启动。即当设备的运行状态发生改变，导致一部分电气元件和电气设备不带电或恢复带电时，可实时分析各设备的带电状态；

c) 应能根据各类规则校验实时数据的正确性，辨识可疑量测，如遥测有值遥信为分、遥测无值遥信为合、PQI 不一致等。

5.4.9 计算

5.4.9.1 功能要求

公式计算应包括以下功能：

a) 宜支持可自定义计算公式，并可从功能界面上以拖拽方式定义计算操作数；

b) 支持加、减、乘、除、三角、对数、逻辑和条件判断等计算，支持的数据类型、运算符、标准函数和语句如下：
1） 支持整型、实型、字符等数据类型的公式计算；
2） 支持算术运算、逻辑运算、关系运算、选择运算等其它运算；
3） 支持指数、对数、三角、反三角运算、绝对值等标准函数运算；
4） 公式运算支持的表达式语句包括循环语句、条件判断语句等复合语句。

c) 可周期启动或触发启动公式计算，启动周期可调，缺省为 5 秒；

d) 支持不少于 50 个操作数；

e) 公式相互引用时能自动调整各个公式的计算次序，确保计算结果的正确性；

f) 能检测公式中存在的语法错误及直接或间接循环引用，并给出提示；

g) 公式计算支持历史重算功能，并支持单个分量修改后相关公式结果自动计算功能；

h) 应提供常用的计算库，支持以下常用计算：
1） 负载率计算；
2） 变压器档位计算；
3） 功率因数计算；
4） 变电站总有功、总无功计算；
5） 其它自定义的公式计算。

5.4.9.2 界面要求

计算界面应具备以下功能:

a) 公式分类目录树展示；

b) 公式信息编辑，包括公式基本信息、公式串、公式操作数等。

5.4.10 责任区与信息分流

5.4.10.1 功能要求

每台工作站应能分配责任区，该工作站应只负责处理所辖责任区内的信息，功能包括：

a) 实时告警信息窗应只显示本责任区范围内的告警信息，无关的告警信息不出现在该工作站；工作站支持可切换不同责任区，具备设置传动责任区功能，将需要传动的变电站或间隔移入传动责任区，正常监视告警信息与传动告警信息通过责任区分流上送；

b) 应只能查询到本责任区范围内的历史告警；

c) 遥控、封锁、挂牌等人工操作应只对本责任区范围内的对象有效，禁止操作无关对象；

d) 限值修改等数据维护只能对本责任区范围内的对象有效。

5.4.10.2 界面要求

责任区与信息分流界面应具备以下功能:

a) 提供统一的责任区管理操作界面，包含责任区的增加、删除、修改、查询功能；

b) 包含配置责任区的厂站、电压等级、设备等功能；

c) 包含配置责任区与角色、责任区与用户的绑定关系功能。

5.5 系统维护工具

5.5.1 系统运行智能诊断

系统运行智能诊断应具备自动系统资源诊断、数据库状态诊断、进程状态诊断、数据一致性诊断功能，并能对诊断结果形成诊断报告。应具备以下功能:

a) 系统资源诊断应支持以下功能：
   1) 应能对集控系统 CPU 负载率进行监视；
   2) 应能对集控系统内存使用率进行监视；
   3) 应能对集控系统网卡中断及速率进行监视；
   4) 应能对集控系统磁盘空间使用率进行监视。
b) 数据库状态诊断应能监视集控系统数据库连接状态；
c) 进程状态诊断应能监视集控系统关键进程频繁投退情况、CPU 占用率状态；
d) 数据一致性诊断应支持以下功能：

应支持通道数据一致性诊断，能对通道数据偏差百分比进行设置，并按照设置条件进行数据一致性对比。

5.6 监盘操作

5.6.1 人工封锁

应提供以下人工封锁功能：

a) 人工输入的数据包括状态量、模拟量及计算量；
b) 对人工输入数据进行有效性检查。

5.6.2 禁止控制和允许控制

禁止控制功能用于禁止对所选对象进行遥控操作处理，应提供以下禁止控制和允许控制功能：

a) 禁止控制/允许控制能支持设备、间隔、站级操作，且在相应等级一次图中有明显禁控标识；
b) 禁止控制功能和允许控制功能成对提供，当前对象必须在当前对象、所有上级对象都允许控制后方可控制，即设备仅在设备、间隔、站端各侧都允许控制后才进行遥控操作。以间隔与站端为例，间隔禁止及场站禁止逻辑：站端禁止、对应下级间隔禁止，站端解除禁止后，对应下级间隔解除，有禁止操作记录的间隔除外、需另在间隔解除；
c) 对所有的禁止控制和允许控制操作进行存档记录。

5.6.3 告警抑制和解除

告警抑制功能用于抑制所选对象的告警上告警窗，应提供以下功能：

a) 告警抑制和解除功能应成对提供；
b) 告警抑制和解除应支持信号、一次设备、二次设备、间隔、厂站级操作；
c) 所有操作记录应进行存档。

5.6.4 标识牌操作

应提供以下标识牌操作功能：

a) 支持常用的标识牌，包括：
   1) 检修：应对现场处于检修试验的设备设置检修标识牌，应支持告警抑制属性配置功能。挂检修牌时，投入告警抑制功能时被抑制对象产生的告警不上告警窗，未投入抑制功能时产生的告警应上送检修告警窗；
   2) 禁止分闸/禁止合闸：禁止对具有该标识牌的设备进行分闸/合闸操作，挂“禁止分闸/禁止合闸”牌后无法执行操作；
   3) 警告：具有该标识牌的设备遥控操作时应提供相关提示；
   4) 接地线：对于不具备接地刀闸的点挂接地线时，应设置该标识牌，并在操作时检查该标

识牌；
5) 故障：禁止对故障隔离设备进行合闸操作；
6) 常亮：对长期点亮的光字牌可挂此牌，将不纳入光字牌合并计算；
7) 保电：挂牌时可填写保电时段、保电任务，设置保电设备量测限值，挂牌后实时监测保电设备，越限时主动告警，并显示保电时间段、保电任务等相关信息；
8) 注释：对变电站、设备可以直接输入注释文字，以此标示当前变电站、设备的描述信息；
9) 调试：支持对在改造传动的设备、间隔、变电站设置该标识牌，相关告警信号进入调试区单独显示；
10) 缺陷：设备本身存在缺陷，或设备本身无缺陷、但其他设备故障原因引起的本设备异常。

b) 提供自定义标识牌功能，仅做名称标识；
c) 能通过人机界面对一个对象设置标识牌或清除标识牌；
d) 支持在远方控制操作时自动检查提示操作对象的标识牌功能；
e) 单个对象能设置多个不同类型标识牌；
f) 支持对多个设备批量挂牌功能；
g) 标识牌操作保存到标识牌一览表中，包括时间、变电站、设备名、标识牌类型、操作员身份和注释等内容，并存档记录；
h) 支持光标放置在注释标识牌上时，自动弹出提示框来展现挂牌时填写的注释信息。

5.7 控制校验

集控系统针对系统中每一个遥控操作指令都需进行操作校验，具体功能应符合如下要求：
a) 应能校验设备投退、闭锁等信息，包括设备在控等进行操作校验；
b) 应能校验设备的操作挂牌信息，包括设备检修、禁止遥控等进行操作校验；
c) 应支持同一时间、同一厂站只有一个主设备可执行遥控操作；
d) 应支持返回错误校验码以及错误原因。

6 系统运行要求

6.1 性能要求

6.1.1 总体要求

系统运行性能总体指标应符合以下要求:
a) 正常状态下：在任意 5 分钟内，服务器 CPU 的平均负荷率≤15%，人机工作站 CPU 的平均负荷率≤20%，主站局域网的平均负荷率≤20%；
b) 事故状态下：在任意 30 秒内，服务器 CPU 的平均负荷率≤30%，人机工作站 CPU 的平均负荷率≤40%，主站局域网的平均负荷率≤30%；
c) 历史数据存储时间跨度≥3 年。

6.1.2 主要性能指标

系统运行主要性能指标应符合以下要求:
a) 实时数据到达集控系统数据采集设备后至实时数据库时间不大于 1 秒；
b) 遥信变化信息到达集控系统数据采集设备后至告警信息推出时间不大于 1 秒；
c) 遥调、遥控量从选中到命令送出系统时间不大于 1 秒；
d) 事故判定后自动推画面时间不大于 3 秒；
e) 画面调用响应时间，主设备相关画面不大于 3 秒，辅助设备相关画面不大于 5s；
f) 系统节点及应用管理：
1) 系统中支持监视的信息个数不小于 200 个；
2) 接口调用响应迅速，不大于 500 毫秒时间内完成。
g) 进程管理：
1) 单节点进程管理支持监视的进程数量不小于 500 个；
2) 接口调用响应迅速，不大于 500 毫秒时间内完成。

h) 系统管理交互界面性能要求：采集数据变化到界面工具展示时间不大于 5 秒；
i) 时钟管理性能要求：各节点机对时误差小于 10 毫秒；
j) 多态多应用性能要求：系统中支持的态不少于 4 个；
k) 历史数据存储：单表存储上限不小于 1000 万；
l) 历史数据查询：在 100 万条记录规模下，日曲线查询服务平均不超过 1s；
m) 模型管理：
   1) 10000 条记录的表，浏览界面打开时间要求≤3 秒；
   2) 10000 条记录的查询≤1 秒，不含触发器、10000 条记录的表，增删改 1 条记录≤1 秒。
n) 实时数据库：
   1) 支持单表存储上限记录数大于 200 万条、支持单表存储空间大于 2G；
   2) 在记录数 200 万的规模下，本地接口查询率 QPS 不小于 100000 次/秒；
   3) 单个实例并发访问数量不低于 200 个。
o) 消息总线：
   1) 千兆带宽下消息总线发送 1KB 消息不低于 10000 个/秒；
   2) 消息总线应保证不丢包；
   3) 任意节点故障，不影响其他节点之间的通信。
p) 服务总线
   基本功能：
   1) 服务总线应保证不丢包；
   2) 服务总线支持服务端并发数：最少 1000 个。
   服务代理：
   1) 服务代理转发数据应保证不丢包；
   2) 服务代理支持广域服务访问并发数，最少 1000 个。
q) 公共服务：
   1) 文件服务：千兆网络下，文件服务应符合：读取文件速度不小于 50MB/秒，写文件速度不小于 40MB/秒；
   2) 文语服务：支持多个应用并发处理且生成语音文件延时 ≤ 1 秒（第三方语音服务指标）。
r) 人机服务：
   1) 数据刷新服务：最快支持 1s 的刷新周期；
   2) 支持不小于 300 个人机并发访问。
s) 人机工具：
   1) 画面编辑器工具：启动画面编辑界面时间，不超过 15 秒；
   2) 画面浏览器工具：启动画面浏览界面时间，不超过 15 秒；画面显示响应时间，90％的画面不超过 2 秒，其它画面不超过 4 秒；右键菜单弹出时间，不超过 1 秒；缩放响应时间<= 1 秒；
   3) 告警监控窗：支持并发处理、支持大数量处理，满足压力测试指标。
t) 图、模维护：
   1) SCD 文件解析加载后，与数据库增量比较时间不超过 180 秒，RCD 文件增量比较不超过 120 秒；
   2) 单厂站 CIM/E 模型导入时间不超过 300 秒，单个 CIM/G 图形导入不超过 60 秒。
u) 数据采集：
   1) 数据采集功能基本容量指标应符合以下要求：遥测量不低于 200000 个；遥信量不低于 800000 个；遥控量不低于 100000 个；遥调量不低于 10000 个；并发遥控数不低于 200 个；厂站接入数不低于 400 个，通道接入数不低于 2048 个；
   2) 遥控命令传送时间：从人机工作站发出控制指令到数据采集服务器端口不大于 1 秒；
   3) 操作控制界面，链路启停、切换生效不大于 2 秒。
v) 数据处理：
   1) 主设备模拟量处理≥20000 个/秒；

2) 主设备状态量处理≥16000 个/秒。

w) 系统安全体系：

1) 用户管理：支持系统并发登录人员不低于 30 人；

2) 权限管理：千兆带宽环境下，身份鉴别、权限认证应该在 100ms 内验证完成；

3) 审计管理：审计日志存储时间跨度不低于 6 个月。

6.1.3 系统备用性能指标

系统备用性能指标应符合以下要求:

a) 人机工作站切换时间≤5 秒；

b) 备用全年可用率≥99.5%。

6.2 可靠性要求

系统可靠性指标应符合以下要求:

a) 系统全年可用率不低于 99.9%；

b) 应用故障切换时间不大于 5 秒；

c) 系统时间与标准时间的误差<10 毫秒；

d) 关键设备平均无故障时间 MTBF＞20000 小时；

e) 由于偶发性故障而发生自动热启动的平均次数＜1 次/2400 小时。

7 接口要求

集控系统与业务中台的数据交互应符合《新一代变电站集中监控系统系列规范 第 9 部分 电网资源业务中台交互》的要求。

附 录 A
（资料性）
基础平台 API 接口定义

A.1 基本原则

基础平台是变电站集中监控系统开发和运行的基础，应具有良好的开放性，提供一个安全、统一、标准及高可用率的应用开发与运行环境，提供标准的 API，支持第三方应用的集成，集控系统基于平台提供系统管理服务、历史数据服务、实时数据服务、消息总线、服务总线、文件服务、日志服务、告警服务、权限服务等多种服务功能，提供统一的基础平台公共数据访问接口。同时基础平台提供了多态多应用管理，应用向平台申请应用号及应用名，根据业务需求集成。附录 A 接口传输内容均采用 GB18030 编码格式。

A.2 基础说明

A.2.1 多态定义

平台为应用提供多态的运行环境，统一了多态的定义，应用可根据业务需求，开展应用多态的集成，态统一定义描述见表 A.1。

表 A.1 态定义

态名称	态实例号	英文名
实时态	1	realtime
预演态	2	study
测试态	4	test
反演态	5	pdr

A.2.2 应用定义

应用是由一组互相紧密关联的功能组成，用于完成某一方面的业务工作，应建立在统一的基础平台上，基于统一的模型、数据及人机工具完成业务需求，应用与平台间、应用与应用之间的数据交换通过平台提供的各类服务接口实现，平台对应用进行统一的管理。态定义基础应用定义见表 A.2。

表 A.2 态定义基础应用定义

应用名	中文描述	应用号
scada	数据处理	100000
public	公共基础应用	1600000
fes	数据采集	400000

A.2.3 数据字典

A.2.3.1 概述

基础平台对应用存储的模型数据表进行统一分配、统一存储、集中管理。平台提供统一的数据字典，供应用查阅，基于平台实时库接口，根据需求裁剪获取需要的模型数据。同时应用根据业务需求向平台申请表号构建自己的模型数据储存结构，平台应支撑应用数据结构在商用库、实时库维护与使用，设备相关测点的值和质量码，应更新设备模型表对应属性。

表 A.3 总表

分类	基础模型设备	英文名称	宏定义	备注
通用字典类型	表信息	sys_table_info	SYS_TABLE_INFO_NO	—
	域信息	sys_column_info	SYS_COLUMN_INFO_NO	—
	菜单定义	sys_menu_info	SYS_MENU_INFO_NO	—
	标志牌定义	token_define	TOKEN_DEFINE_NO	—
容器类	区域表	subcontrolarea	SUBCONTROLAREA_NO	—
	电压类型	basevoltage	BASEVOLTAGE_NO	—
	厂站类	substation	SUBSTATION_NO	—
	间隔类	bay	BAY_NO	—
	变电站区域	substation_area	SUBSTATION_AREA_NO	—
	变电站屏柜	substation_cabinet	SUBSTATION_CABINET_NO	—
	责任区类	resp_area_def	RESP_AREA_DEF_NO	—
主设备模型	断路器	breaker	BREAKER_NO	—
	刀闸（隔离开关）	disconnector	DISCONNECTOR_NO	—
	线路	acline	ACLINE_NO	—
	负荷	energyconsumer	ENERGYCONSUMER_NO	—
	变压器	powertransformer	POWERTRANSFORMER_NO	—
	变压器绕组	transformerwinding	TRANSFORMERWINDING_NO	—
	接地刀闸	grounddisconnector	GROUNDDISCONNECTOR_NO	—
	并联补偿器	compensator_p	COMPENSATOR_P_NO	—
	串联补偿器	compensator_s	COMPENSATOR_S_NO	—
	交流线段	aclinesegment	ACLINESEGMENT_NO	—
	线段端点	aclineend	ACLINEEND_NO	—
	母线	busbarsection	BUSBARSECTION_NO	—
	电流互感器	transformer_c	TRANSFORMER_C_NO	—
	电压互感器	transformer_v	TRANSFORMER_V_NO	—
	消弧线圈	arcsuppressioncoil	ARCSUPPRESSIONCOIL_NO	—
	避雷器	lightningarrester	LIGHTNINGARRESTER_NO	—
	保护信号	relaysig	RELAYSIG_NO	—
	其他遥信	signal	SIGNAL_NO	—
	其他遥测	measure	MEASURE_NO	—
	主设备遥测	measanalog	MEASANALOG_NO	主设备遥测测点汇总表
	主设备遥信	measpoint	MEASPOINT_NO	主设备遥信测点汇总表
二次设备类	二次基本信息表	cntrl_ied	CNTRL_IED_NO	—
	二次定值区号表	cntrl_ied_ldevice	CNTRL_IED_LDEVICE_NO	—
防误类	网门表	electriccub	ELECTRICCUB_NO	—
	接地桩	groundstake	GROUNDSTAKE_NO	—
辅助设备	辅助设备	auxiliary_device	AUXILIARY_DEVICE_NO	—
	辅助设备遥测	auxiliary_yc	AUXILIARY_YC_NO	—
	辅助设备遥信	auxiliary_yx	AUXILIARY_YX_NO	—
控制参数类	主设备遥控	controldigital	CONTROLDIGITAL_NO	—
	主设备设点	controlsetpnt	CONTROLSETPNT_NO	—
	档位升降控制	controltap	CONTROLTAP_NO	—
	辅设备遥控	auxiliary_controldigital	AUXILIARY_CONTROLDIGITAL_NO	—
	辅设备设点	auxiliary_controlsetpnt	AUXILIARY_CONTROLSETPNT_NO	—

前置信息类	前置主设备遥信信息	fes_yx_define	FES_YX_DEFINE_NO	—
	前置主设备遥测信息	fes_yc_define	FES_YC_DEFINE_NO	—
	下行主设备遥控信息	send_dc	SEND_DC_NO	—
	下行主设备档位遥调	send_pd	SEND_PD_NO	—
	下行主设备设点信息	send_sp	SEND_SP_NO	—
	前置辅设备遥信信息	fes_yx_define_aem	FES_YX_DEFINE_AEM_NO	—
	前置辅设备遥测信息	fes_yc_define_aem	FES_YC_DEFINE_AEM_NO	—
	下行辅设备遥控信息	send_dc_aem	SEND_DC_AEM_NO	—
	下行辅设备设点信息	send_sp_aem	SEND_SP_AEM_NO	—
操作结果类	人工操作结果	op_info	OP_INFO_NO	—
	主设备限值表	limitsets	LIMITSETS_NO	—
	标志牌操作结果	token_info	TOKEN_INFO_NO	—
	对端代结果表	replace_info	REPLACE_INFO_NO	—
	旁路代结果表	bypass_result	BYPASS_RESULT_NO	—
历史告警类	遥信变位	yx_bw	YX_BW_NO	—
	遥测越限	yc_over	YC_OVER_NO	—
	SOE	yx_soe	YX_SOE_NO	—
	主设备操作	op_ctrl	OP_CTRL_NO	—
	辅设备操作	op_ctrl_aux	OP_CTRL_AUX_NO	—
	二次设备操作	relay_op	RELAY_OP_NO	—
	注：可通过 A.5.11、A.5.12 获取表号与表名之间的转换关系。			

A.2.3.2 通用字典类

表 A.4 表信息（sys_table_info）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度	备注
1	表号	table_id	表号	int	4	—
2	表英文名称	table_name_eng	表英文名称	string	32	—
3	表中文名称	table_name_chn	表中文名称	string	64	—

表 A.5 域信息（sys_column_info）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度	备注
1	表号	table_id	表号	int	4	—
2	域号	column_id	域号	short	2	—
3	域英文名称	column_name_eng	域英文名称	string	32	—
4	域中文名称	column_name_chn	域中文名称	string	64	—
5	数据类型	data_type	域数据类型	uchar	1	—
6	数据长度	data_length	域数据长度	short	2	—
7	菜单名	menu_name	域引用的菜单	string	40	如有菜单引用，根据菜单名获取 A 表菜单定义表中对应菜单名的记录，实际表中引用显示的为菜单的显示值，存储的为菜单的实际值。

表 A.6 菜单定义（sys_menu_info）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度	备注
1	菜单名	menu_name	菜单名	string	40	—
2	菜单序号	menu_no	菜单序号	uchar	1	—
3	实际值	actual_value	实际值	int	4	菜单存储的实际值
4	显示值	display_value	显示值	string	40	菜单显示的内容

表 A.7 标志牌定义（token_define）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度	备注
1	标识	token_no	标志牌唯一标识	long	8	—
2	中文名称	name	标志牌名称	string	64	—
3	标志牌类型	token_type	标志牌类型	uchar	1	引用菜单“标志牌类型”，根据菜单名从 A.菜单定义中查询详细的类型分类
4	图元名称	icon_name	标志牌图元名称	string	64	—

A.2.3.3 容器类

表 A.8 电压类型类（basevoltage）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	在整个模型文件中唯一，系统中的标识	long	8
2	中文名称	name	如 500kV，220kV，1000kV，35kV,其中 k 小写，V 大写。见《电网设备通用数据模型命名规范（试行）》的电压等级命名。	string	64
3	基准电压	nomvol	小数点后面保留两位有效位数	float	4

表 A.9 区域类（subcontrolarea）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	为原系统中的标识，在整个模型文件中唯一	Long	8
2	中文名称	name	—	String	64
3	父区域 ID	father_id	—	Long	8
(1) 区域细化到地区，通过父区域标识关联建立区域间的隶属关系。 (2) 字段类型：s 表示字符串，f 表示浮点数，i 表示整数，l 表示 8 位长整型，下同。					

表 A.10 厂站类(substation)

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	原系统中的标识，在整个模型文件中唯一	long	8
2	中文名称	name	原系统中的设备中文名称	string	64
3	调度编号	code	调度编号	string	32
4	厂站类型	st_type	变电站类型，引用“厂站	int	4

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
			类型”菜单，菜单值：4 变电站、5 虚拟站		
5	所属区域标识	subarea_id	对区域标识的引用对应 subcontrolarea 的 id	long	8
6	最高电压类型 id	bv_id	对基准电压标识的引用对应 basevoltage 的 id	long	8
7	责任区	resp_area	责任区值	long	8
8	图形名称	graph_name	主接线图图形名称	string	64

表 A.11 间隔类（bay）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	系统中的标识，整个模型文件中唯一	long	8
2	中文名称	name	标准带路径全名	string	64
2	调度编号	code	调度编号	string	32
3	所属厂站标识	st_id	厂站标识的引用。	long	8
4	所属电压等级标识	bv_id	电压等级标识的引用。	long	8
5	责任区	resp_area	责任区值	long	8
6	间隔分图	graph_name	间隔分图	string	64

表 A.12 变电站区域（substation_area）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	系统中的标识，整个模型文件中唯一	long	8
2	中文名称	area_name	变电站区域名称	string	64
3	变电站 ID	st_id	厂站标识的引用	long	8
4	父区域 ID	father_id	引用变电站区域	long	8
5	责任值	resp_area	责任区值	long	8

表 A.13 变电站屏柜（substation_cabinet）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	系统中的标识，整个模型文件中唯一	long	8
2	中文名称	name	屏柜名称	string	64
3	变电站 ID	st_id	厂站标识的引用	long	8
4	间隔 ID	bay_id	间隔标识引用	long	8
5	变电站区域 ID	substation_area	变电站区域的引用	long	8

表 A.14 责任区定义（resp_area_def）

序号	属性项	属性项英文名	域中文名	数据类型	数据长度
1	责任区 ID	area_id	责任区 ID	long	8
2	责任区名称	area_name	责任区名称	string	64
3	责任区序号	serial_no	责任区序号	int	4
4	责任区类型	area_type	责任区类	int	4

			型，0代表基本责任区、1复合责任区		
5	责任区值	area_value	责任区值	long	8
注：责任区值，转换为二进制，每一位代表一个基本责任区，多个基本责任区组合为复合责任区					

A.2.3.4 主设备模型

表 A.15 断路器类(breaker)

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	在整个模型文件中唯一，系统中的标识	long	8
2	中文名称	name	标准带路径全名	string	64
3	调度编号	code	调度编号属性	string	32
4	断路器类型	brk_type	引用“断路器类型”菜单	uchar	1
5	首端连接节点号	ind	首端连接节点号	long	8
6	末端连接节点号	jnd	末端连接节点号	long	8
7	所属厂站标识	st_id	对厂站标识的引用	long	8
8	所属间隔	bay_id	对间隔标识的引用	long	8
9	基准电压标识	bv_id	对基准电压标识的引用	long	8
10	遥信值	point	开关分合位置	uchar	1
11	状态	status	遥信质量码	int	4
12	拓扑着色	tpcolor	拓扑分析结果	uchar	1
13	责任区	resp_area	责任区值	long	8
14	设备健康评分	health_score	设备健康评分	float	4
15	设备健康等级	health_level	设备健康等级，引用“设备诊断状态等级”，0未评价，1正常，2警告，3，异常，4严重	int	4
16	功率因数	pw_factor	功率因数	float	4
17	功率因数状态	pw_factor_qual	功率因数状态	int	4
注：“拓扑着色”，126可疑接地，127接地；1-125都是正常带电岛，按照电气岛中逻辑母线数量从多到少排，最多的是1；0是停电，下同。					

表 A.16 刀闸类（disconnector）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	在整个模型文件中唯一，系统中的标识	long	8
2	中文名称	name	标准带路径全名	string	64
3	调度编号	code	调度编号	string	32
4	首端连接节点号	ind	首端连接节点号	long	8
5	末端连接节点号	jnd	末端连接节点号	long	8
6	所属厂站标识	st_id	对厂站标识的引用	long	8
7	所属间隔	bay_id	对间隔标识的引用	long	8
8	基准电压标识	bv_id	对基准电压标识的引用	long	8
9	遥信值	point	刀闸分合位置	uchar	1
10	状态	status	遥信质量码	int	4
11	拓扑着色	tpcolor	拓扑分析结果	uchar	1
12	责任区	resp_area	责任区值	long	8
13	设备健康评分	health_score	设备健康评分	float	4
14	设备健康等级	health_level	设备健康等级，引	int	4

			用“设备诊断状态等级”，0 未评价，1 正常，2 警告，3，异常，4 严重		

表 A.17 线路类(acline)

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	在整个模型文件中唯一，系统中的标识	long	8
2	中文名称	name	标准带路径全名	string	64
3	调度编号	code	调度编号	string	32
4	基准电压标识	bv_id	对基准电压标识的引用	long	8
说明：线路类是个容器类，一个线路对象可以包含多个交流线段，主要用于 T 接线的描述。					

表 A.18 负荷类(energyconsumer)

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	系统中的标识，在整个模型文件中唯一	long	8
2	中文名	name	标准带路径全名	string	64
3	调度编号	code	调度编号	string	32
4	厂站标识	st_id	对厂站标识的引用	long	8
5	所属间隔	bay_id	对间隔标识的引用	long	8
6	物理连接节点	nd	物理连接节点	long	8
7	基准电压标识	bv_id	对基准电压标识的引用	long	8
8	有功值	p	有功值	float	4
9	有功质量码	p_qual	有功质量码	int	4
10	无功值	q	无功值	float	4
11	无功质量码	q_qual	无功质量码	int	4
12	电流值	i	电流值	float	4
13	电流质量码	i_qual	电流质量码	int	4
14	拓扑着色	tpcolor	拓扑分析结果	uchar	1
15	责任区	resp_area	责任区值	long	8
16	负载率	load_rate	负载率	float	4
17	负载率质量码	load_rate_qual	负载率质量码	int	4
18	功率因数	pw_factor	功率因数	float	4
19	功率因素状态	pw_factor_qual	功率因素状态	int	4

表 A.19 变压器类 (powertransformer)

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	系统中的标识，在整个模型文件中唯一。	long	8
2	中文名称	name	标准带路径全名	string	64
3	调度编号	code	调度编号	string	32
4	类型	tr_type	引用“变压器类型”菜单	uchar	1
5	厂站标识	st_id	对厂站标识的引用	long	8

6	所属间隔	bay_id	对间隔标识的引用	long	8
7	拓扑着色	tpcolor	拓扑分析结果	uchar	1
8	绕组类型	wind_type	变压器绕组类型，引用“绕组类型”菜单	uchar	1
9	责任区	resp_area	责任区值	long	8
10	设备健康评分	health_score	设备健康评分	float	4
11	设备健康等级	health_level	设备健康等级，引用“设备诊断状态等级”，0 未评价，1 正常，2 警告，3，异常，4 严重	int	4

表 A.20 变压器绕组类（transformerwinding）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	系统中的标识，在整个模型文件中唯一	long	8
2	中文名称	name	标准带路径全名	string	64
3	调度编号	code	调度编号	string	32
4	绕组类型	wind_type	高/中/低	uchar	1
5	厂站标识	st_id	对厂站标识的引用	long	8
6	所属间隔	bay_id	对间隔标识的引用	long	8
7	变压器标识	tr_id	对变压器标识的引用	long	8
8	物理连接节点	nd	物理连接节点	long	8
9	基准电压标识	bv_id	对基准电压标识的引用	long	8
10	有功值	p	有功	float	4
11	有功值质量码	p_qual	有功值质量码	int	4
12	无功值	q	无功	float	4
13	无功值质量码	q_qual	无功值质量码	int	4
14	电流值	i	电流	float	4
15	电流值质量码	i_qual	电流值质量码	int	4
16	分接头位置	tap	档位	float	4
17	分接头质量码	tap_qual	分接头质量码	int	4
18	拓扑着色	tpcolor	拓扑分析结果	uchar	1
19	中性点连接点号	knd	中性点连接点号	long	8
20	责任区	resp_area	责任区值	long	8
21	A 相电流幅值	i_a_value	A 相电流幅值	float	4
22	A 相电流质量码	i_a_qual	A 相电流质量码	int	4
23	B 相电流幅值	i_b_value	B 相电流幅值	float	4
24	B 相电流质量码	i_b_qual	B 相电流质量码	int	4
25	C 相电流幅值	i_c_value	C 相电流幅值	float	4
26	C 相电流质量码	i_c_qual	C 相电流质量码	int	4
27	负载率	load_rate	负载率	float	4
28	负载率质量码	load_rate_qual	负载率质量码	int	4
29	功率因数	pw_factor	功率因数	float	4
30	功率因数状态	pw_factor_qual	功率因数状态	int	4

表 A.21 接地刀闸（grounddisconnector）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	系统中关于接地刀闸	long	8

			设备标识，在整个系统模型中唯一		
2	中文名称	name	系统中关于接地刀闸设备中文名称描述，如“201-7 接地刀闸”。	string	64
3	调度编号	code	调度编号	string	32
4	变电站 ID	st_id	关于厂站表的引用，设备所属厂站	long	8
5	间隔 ID	bay_id	关于间隔表的引用，设备所属间隔	long	8
6	电压类型 ID	bv_id	对基准电压表的引用，设备的电压类型	long	8
7	连接点号	nd	拓扑连接点号	long	8
8	遥信值	point	设备遥信值	uchar	1
9	状态	status	遥信值质量码	int	4
10	拓扑着色	tpcolor	拓扑分析结果	uchar	1
11	责任区	resp_area	责任区值	long	8
12	设备健康评分	health_score	设备健康评分	float	4
13	设备健康等级	health_level	设备健康等级，引用“设备诊断状态等级”，0 未评价，1 正常，2 警告，3，异常，4 严重	int	4

表 A.22 并联补偿器类（compensator_p）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	系统中并联补偿器设备标识，在整个系统模型中唯一	long	8
2	中文名称	name	系统中设备中文名称	string	64
3	调度编号	code	调度编号	code	32
4	变电站 ID	st_id	关于厂站表的引用，设备所属厂站	long	8
5	间隔 ID	bay_id	关于间隔表的引用，设备所属间隔	long	8
6	电压类型 ID	bv_id	对基准电压表的引用，设备的电压类型	long	8
7	连接点号	nd	拓扑连接点号	long	8
8	补偿器类型	cp_type	并联电容，并联电抗	uchar	1
9	无功值	q	设备无功值	float	4
10	无功质量码	q_qual	无功值质量码	int	4
11	A 相电流值	i_a_value	设备 A 相电流值	float	4
12	A 相电流质量码	i_a_qual	A 相电流值质量码	int	4
13	B 相电流值	i_b_value	设备 B 相电流值	float	4
14	B 相电流质量码	i_b_qual	B 相电流值质量码	int	4
15	C 相电流值	i_c_value	设备 C 相电流值	float	4
16	C 相电流质量码	i_c_qual	B 相电流值质量码	int	4
17	拓扑着色	tpcolor	拓扑分析结果	uchar	1

18	责任区	resp_area	责任区值	long	8
19	设备健康评分	health_score	设备健康评分	float	4
20	设备健康等级	health_level	设备健康等级，引用“设备诊断状态等级”，0 未评价，1 正常，2 警告，3，异常，4 严重	int	4

表 A.23 串联补偿器类（compensator_s）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	系统中并联补偿器设备标识，在整个系统模型中唯一	long	8
2	中文名称	name	系统中设备中文名称	string	64
3	调度编号	code	调度编号	string	32
4	变电站 ID	st_id	关于厂站表的引用，设备所属厂站	long	8
5	间隔 ID	bay_id	关于间隔表的引用，设备所属间隔	long	8
6	电压类型 ID	bv_id	对基准电压表的引用，设备的电压类型	long	8
7	首端连接点号	ind	拓扑连接点号	long	8
8	末端连接点号	jnd	拓扑连接点号	long	8
9	拓扑着色	tpcolor	拓扑分析结果	uchar	1
10	责任区	resp_area	责任区值	long	8
11	设备健康评分	health_score	设备健康评分	float	4
12	设备健康等级	health_level	设备健康等级，引用“设备诊断状态等级”，0 未评价，1 正常，2 警告，3，异常，4 严重	int	4

表 A.24 交流线段(aclinesegment)

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	系统中关于交流线段设备标识，在整个系统模型中唯一	long	8
2	中文名称	name	交流线段在系统中的中文名称。	string	64
3	调度编号	code	调度编号	string	32
4	一端变电站 ID	ist_id	交流线段一端厂站，对厂站表的引用	long	8
5	二端变电站 ID	jst_id	交流线段二端厂站，对厂站表的引用	long	8
6	电压类型 ID	bv_id	对基准电压表的引用，设备的电压等类型	long	8
7	首端连接节点号	ind	首端连接节点号	long	8
8	末端连接节点号	jnd	末端连接节点号	long	8
9	拓扑着色	tpcolor	拓扑分析结果	uchar	1

表 A.25 交流线端点(aclineend)

序号	属性项	属性项英文名	属性要求	数据类型	数据长度

1	标识	id	系统中关于交流线段端点设备标识，在整个系统模型中唯一	long	8
2	中文名称	name	系统中的中文名称。如葛玉线	string	64
3	调度编号	code	调度编号	string	32
4	变电站 ID	st_id	关于厂站表的引用，设备所属厂站	long	8
5	间隔 ID	bay_id	关于间隔表的引用，设备所属间隔	long	8
6	线段 ID	aclnseg_id	关于交流线段表的引用，设备所属交流线段	long	8
7	电压类型 ID	bv_id	对基准电压表的引用，设备的电压类型	long	8
8	连接点号	nd	拓扑连接点号	long	8
9	有功值	p	设备的有功值	float	4
10	有功质量码	p_qual	有功值质量码	int	4
11	无功值	q	设备的无功值	float	4
12	无功质量码	q_qual	无功值质量码	int	4
13	电流值	i	设备的电流值	float	4
14	电流质量码	i_qual	电流值质量码	int	4
15	拓扑着色	tpcolor	拓扑分析结果	uchar	1
16	责任区	resp_area	责任区值	long	8
17	负载率	load_rate	负载率	float	4
18	负载率质量码	load_rate_qual	负载率质量码	int	4

表 A.26 母线（busbarsection）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	系统中关于母线设备的标识，在整个系统模型中唯一	long	8
2	中文名称	name	系统中的中文名称。	string	64
3	调度编号	code	调度编号	string	32
4	变电站 ID	st_id	关于厂站表的引用，设备所属厂站	long	8
5	间隔 ID	bay_id	关于间隔表的引用，设备所属间隔	long	8
6	电压类型 ID	bv_id	对基准电压表的引用，设备的电压类型	long	8
7	母线类型	bs_type	引用“母线类型”菜单，1 主母、2 副母、3 I 段母线、4II 段母线、5 虚拟母线，可根据实际情况扩展	int	4
8	连接点号	nd	拓扑连接点号	long	8
9	线电压	v	线电压	float	4
10	线电压质量码	v_qual	线电压质量码	int	4
11	相角	ang	相角	float	4
12	相角质量码	ang_qual	相角质量码	int	4
13	频率	f	频率	float	4
14	频率质量码	f_qual	频率质量码	int	4

15	拓扑着色	tpcolor	拓扑分析结果	uchar	1
16	责任区	resp_area	责任区值	long	8
17	设备健康评分	health_score	设备健康评分	float	4
18	设备健康等级	health_level	设备健康等级，引用“设备诊断状态等级”，0 未评价，1 正常，2 警告，3，异常，4 严重	int	4

表 A.27 电流互感器（transformer_c）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	ID	id	在整个系统模型中唯一	long	8
2	厂站 id	st_id	关于厂站表的引用，设备所属厂站	long	8
3	调度编号	code	调度编号	string	32
4	中文名称	name	中文名称	string	64
5	间隔 id	bay_id	关于间隔表的引用，设备所属间隔	long	8
6	电压类型 id	bv_id	对基准电压表的引用，设备的电压类型	long	8
7	拓扑着色	tpcolor	拓扑着色	uchar	1
8	首端连接点号	ind	首端连接点号	long	8
9	末端连接点号	jnd	末端连接点号	long	8
10	责任区	resp_area	责任区值	long	8
11	设备健康评分	health_score	设备健康评分	float	4
12	设备健康等级	health_level	设备健康等级，引用“设备诊断状态等级”，0 未评价，1 正常，2 警告，3，异常，4 严重	int	4

表 A.28 电压互感器（transformer_v）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	ID	id	在整个系统模型中唯一	long	8
2	厂站 id	st_id	关于厂站表的引用，设备所属厂站	long	8
3	调度编号	code	调度编号	string	32
4	中文名称	name	中文名称	string	64
5	间隔 id	bay_id	关于间隔表的引用，设备所属间隔	long	8

6	电压类型 id	bv_id	对基准电压表的引用，设备的电压类型	long	8
7	连接点号	nd	连接点号	long	8
8	拓扑着色	tpcolor	拓扑着色	uchar	1
9	责任区 ID	resp_area	责任区值	long	8
10	设备健康评分	health_score	设备健康评分	float	4
11	设备健康等级	health_level	设备健康等级，引用“设备诊断状态等级”，0 未评价，1 正常，2 警告，3，异常，4 严重	int	4

表 A.29 消弧线圈（arcsuppressioncoil）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	系统中关于消弧线圈设备的标识，在整个系统模型中唯一	long	8
2	中文名称	name	系统中的中文名称。	string	64
3	调度编号	code	调度编号	code	
4	变电站 ID	st_id	关于厂站表的引用，设备所属厂站	long	8
5	间隔 ID	bay_id	关于间隔表的引用，设备所属间隔	long	8
6	电压类型 ID	bv_id	对基准电压表的引用，设备的电压类型	long	8
7	连接点号	nd	拓扑连接点号	long	8
8	拓扑着色	tpcolor	拓扑分析结果	uchar	1
9	责任区	resp_area	责任区值	long	8
10	设备健康评分	health_score	设备健康评分	float	4
11	设备健康等级	health_level	设备健康等级，引用“设备诊断状态等级”，0 未评价，1 正常，2 警告，3，异常，4 严重	int	4

表 A.30 避雷器（lightningarrester）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	系统中关于劈雷器的标识，在整个系统模型中唯一	long	8
2	中文名称	name	系统中的中文名称。	string	64
3	变电站 ID	st_id	关于厂站表的引用，设备所属厂站	long	8
4	间隔 ID	bay_id	关于间隔表的引用，设备所属间隔	long	8

5	电压类型 ID	bv_id	对基准电压表的引用，设备的电压类型	long	8
6	连接点号	nd	拓扑连接点号	long	8
7	拓扑着色	tpcolor	拓扑分析结果	uchar	1
8	责任区	resp_area	责任区值	long	8
9	设备健康评分	health_score	设备健康评分	float	4
10	设备健康等级	health_level	设备健康等级，引用“设备诊断状态等级”，0 未评价，1 正常，2 警告，3，异常，4 严重	int	4

表 A.31 保护信号表（relaysig）

序号	属性项	属性项英文名	属性要求	数据长度	数据类型
1	标识	id	系统中对应保护信号 ID，整个系统模型中唯一	long	8
2	中文名称	name	系统中的中文名字	string	128
3	变电站 ID	st_id	对厂站表的引用	long	8
4	间隔 ID	bay_id	对间隔表的引用	long	8
5	值	value	保护信号值	uchar	1
6	质量码	qual	质量码	int	4
7	责任区	resp_area	责任区值	long	8
8	是光字牌	if_light	是否参加光字牌计算	uchar	1
9	是否常亮	if_light_on	是否常亮	uchar	1
10	光字牌值	light_value	光字牌值	uchar	1
11	光字牌质量码	light_qual	光字牌质量码	int	4
12	动作时间	on_time	动作时间	long	8
13	复归时间	off_time	复归时间	long	8
14	主设备告警等级	maindev_alarm_level	主设备告警等级，光字牌等级计算用	int	4
15	主设备类型	belong_dev_type	主设备类型，光字牌分类计算用	int	4
16	所属设备	relaydev_id	所属设备 ID	long	8

表 A.32 其他遥信（signal）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	在整个模型文件中唯一，系统中的标识	long	8
2	中文名称	name	标准带路径全名	string	64
3	所属厂站标识	st_id	对厂站标识的引用	long	8
4	所属间隔	bay_id	对间隔标识的引用	long	8
5	基准电压标识	bv_id	对基准电压标识的引用	long	8
6	遥信值	value	遥信值	uchar	1
7	状态	qual	质量码	int	4
8	责任区	resp_area	责任区值	long	8

表 A.33 其他遥测（measure）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	在整个模型文件中唯一，系统中的标识	long	8
2	中文名称	name	标准带路径全名	string	64
3	所属厂站标识	st_id	对厂站标识的引用	long	8
4	所属间隔	bay_id	对间隔标识的引用	long	8
5	基准电压标识	bv_id	对基准电压标识的引用	long	8
6	遥测值	value	遥测值	float	4
7	状态	qual	质量码	int	4
8	责任区	resp_area	责任区值	long	8

表 A.34 主设备遥测（measanalog）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	测点标识，系统中唯一	long	8
2	遥测 ID	alg_id	系统中对应设备的遥测 ID，整个系统模型中唯一	long	8
3	中文名称	name	系统中的中文名字	string	128
4	变电站 ID	st_id	对厂站表的引用	long	8
5	更新时间	upt_time	遥测值更新时间	long	8
6	变化时间	chg_time	遥测值发生变化时间	long	8
7	reference 属性	reference	reference 属性	string	128
8	信号等级	warn_level	告警等级	int	4
9	合理上限	valid_up	合理上限	float	4
10	合理下限	valid_low	合理下限	float	4
11	信息对象	repository_id	标准监控信息对象 ID，事件化应用用	long	8
注：alg_id 应符合 A2.4 ID 编码要求，在主设备 ID 的基础上含域号信息，设备模型中对应域信息表中的域号，遥测值和质量码应从设备模型表中对应域获取。					

表 A.35 主设备遥信（measpoint）

序号	属性项	属性项英文名	属性要求	数据长度	数据类型
1	标识	id	测点标识，系统中唯一	long	8
2	遥信 ID	pnt_id	系统中对应设备的遥信 ID，整个系统模型中唯一	long	8
3	中文名称	name	系统中的中文名字	string	128
4	变电站 ID	st_id	对厂站表的引用	long	8
5	更新时间	upt_time	遥信值更新时间	long	8
6	变化时间	chg_time	遥信值发生变化时间	long	8
7	reference 属性	reference	reference	string	128
8	信号等级	warn_level	告警等级	int	4

9	信息对象	repository_id	标准监控信息对象ID，事件化应用用	long	8
注：pnt_id 应符合 A2.4 ID 编码要求，在主设备 ID 的基础上含域号信息，设备模型中对应域信息表中的域号，遥信值和质量码应从设备模型表中对应域获取，遥信值 0：遥信分、保护复归，遥信值 1：遥信合、保护动作					

A.2.3.5 辅设备模型

辅助设备模型数据字典见表 A.36。辅助设备遥测模型数据字典见表 A.37。辅助设备遥信模型数据字典见表 A.38。

表 A.36 辅助设备（auxiliary_device）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	系统中关于辅助设备的标识，在整个系统模型中唯一	long	8
2	中文名称	dev_name	系统中关于辅助设备的中文描述	string	64
3	变电站 ID	st_id	关于厂站表的引用，设备所属厂站	long	8
4	设备类别	aux_class	辅助设备类别：引用“辅助设备类别”菜单，包括：动环设备、安防设备、消防设备、在线监测	int	4
5	设备类型	aux_type	与设备类型是父子关系，（1）设备类型为“动环设备”时，引用“辅助类型-动环”菜单；（2）设备类型为“安防设备”时，引用“辅助类型-安防”菜单；（3）设备类型为“消防设备”时，引用“辅助类型-消防”；（4）设备类型为“在线监测”时，引用“辅助类型-监测”。	int	4
6	责任区	resp_area	责任区值	long	8
7	所属站内区域	subarea_id	对站内区域标识的引用，对应 substation_area 表中的 id	long	8

表 A.37 辅助设备遥测（auxiliary_yc）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	系统中辅助设备遥测 ID，整个系统模型中唯一	long	8
2	中文名称	yc_name	辅助设备遥测中文名称	String	128
3	变电站 ID	st_id	关于厂站表的引用，设备量测所属厂站	long	8
4	所属设备 ID	dev_id	关于辅助设备表引用，量测所属设备	long	8
5	遥测值	yc_value	遥测值	float	4
6	质量码	qual	遥测质量码	int	4
7	更新时间	upt_time	遥测值更新时间	long	8
8	变化时间	chg_time	遥测值发生变化时间	long	8
9	参引	reference	参引	string	128

10	遥测类别	yc_type	辅助遥测类别：引用“辅助设备类别”菜单，包括：动环设备、安防设备、消防设备、在线监测	int	4
11	遥测子类型	yc_sub_type	与遥测类型是父子关系，（1）设备类型为“动环设备”时，引用“辅助遥测-动环”菜单；（2）设备类型为“安防设备”时，引用“辅助遥测-安防”菜单；（3）设备类型为“消防设备”时，引用“辅助遥测-消防”；（4）设备类型为“在线监测”时，引用“辅助遥测-监测”。	int	4
12	关联一次设备	primary_dev	关联的一次设备，引用一次设备 ID	long	8
13	责任区	resp_area	责任区值	long	8
14	合理上限	valid_up	合理上限	float	4
15	合理下限	valid_low	合理下限	float	4
16	信号等级	warn_level	信号等级	int	4
17	下限告警值	low_alarm_val	下限告警值	float	4
18	下限预警值	low_warn_val	下限预警值	float	4
19	上限告警值	up_alarm_val	上限告警值	float	4
20	上限预警值	up_warn_val	上限预警值	float	4
21	信息对象	repository_id	标准监控信息对象 ID，事件化应用用	long	8
22	所属二次设备	relay_id	关联的二次设备 ID 信息	long	8

表 A.38 辅助设备遥信（auxiliary_yx）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	系统中辅助设备遥信 ID，整个系统模型中唯一	long	8
2	中文名称	yx_name	辅助设备遥信中文名称	string	128
3	变电站 ID	st_id	关于厂站表的引用，设备量测所属厂站	long	8
4	所属设备 ID	dev_id	关于辅助设备表引用，量测所属设备	long	8
5	遥信值	yx_value	遥信值	uchar	1
6	遥信质量码	qual	遥信质量码	int	4
7	更新时间	upt_time	遥信值更新时间	long	8
8	变化时间	chg_time	遥信值发生变化时间	long	8
9	参引	reference	参引	string	128
10	遥信类别	yx_type	辅助遥信类别：引用“辅助设备类别”菜单，包括：动环设备、安防设备、消防设备、在线监测	int	4
11	遥信子类型	yx_sub_type	与遥信类型是父子关系，（1）设备类型为“动环设备”时，引用“辅助遥信-动环”菜单；（2）设备类型为“安防设备”时，引用“辅助	int	4

			遥信-安防”菜单；（3）设备类型为“消防设备”时，引用“辅助遥信-消防”；（4）设备类型为“在线监测”时，引用“辅助遥信-监测”。		
12	关联一次设备	primary_dev	关联的一次设备，引用一次设备 ID	long	8
13	责任区	resp_area	责任区值	long	8
14	是否光字牌	if_light	是否光字牌	uchar	1
15	光字牌值	light_value	光字牌值	uchar	1
16	光字牌质量码	light_qual	光字牌质量码	int	4
17	是否常亮	if_light_on	是否常亮	uchar	1
18	信号等级	warn_level	信号等级	int	4
19	信息对象	repository_id	标准监控信息对象 ID，事件化应用用	long	8
20	所属二次设备	relay_id	关联的二次设备 ID 信息	long	8

A.2.3.6 二次设备模型

二次设备模型数据字典见表 A.39。二次定制区号数据字典见表 A.40。

表 A.39 二次基本信息表（cntrl_ied）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	系统中的标识，,整个模型文件中唯一	long	8
2	中文名称	name	系统中的中文名称	stirng	128
3	iedName	ied_name	装置名称	stirng	64
4	厂站 ID	st_id	关于厂站表的引用，设备所属厂站	long	8
5	间隔 ID	bay_id	关于间隔表的引用，设备所属间隔	long	8
6	电压类型 ID	vl_id	关于电压类型表的引用，设备所属电压	long	8
7	类型	type	二次设备的类型	int	4
8	型号	model	二次设备的型号	stirng	32
9	103 通讯地址	addr	103 通讯地址	int	4
10	软件版本	soft_version	软件版本	stirng	64

表 A.40 二次定值区号表（cntrl_ied_ldevice）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	系统中对应设备的遥测 ID，整个系统模型中唯一	long	8
2	中文名称	name	系统文名称	stirng	128
3	厂站 ID	st_id	关于厂站表的引用，设备所属厂站	long	8
4	二次设备 ID	ied_id	关于二次设备表的引用，设备所属二次设备	long	8
5	间隔 ID	bay_id	关于间隔表的引用，设备所属间隔	long	8

6	最小定值区号	min_sg	最小定值区号	int	4
7	最大定值区号	max_sg	最大定值区号	int	4
8	当前定值区号	act_sg	当前定值区号	float	4
9	当前定值区号质量码	act_sg_qual	当前定值区号质量码	int	4

A.2.3.7 防误模型

网门类模型数据字典见表 A.41。接地桩类模型数据字典见表 A.42。

表 A.41 网门表（electriccub）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	系统中的标识，整个模型文件中唯一	long	8
2	中文名称	name	标准带路径全名	string	128
	调度编号	code	调度编号	string	32
3	变电站 ID	st_id	厂站标识的引用	long	8
4	间隔 ID	bay_id	间隔标识引用	long	8
5	电压类型 ID	bv_id	电压等级标识的引用。	long	8
6	值	value	网门状态值	uchar	1
7	质量码	qual	质量码	int	4
8	责任区	resp_area	责任区值	long	8

表 A.42 接地桩线（groundstake）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	系统中的标识，整个模型文件中唯一	long	8
2	中文名称	name	标准带路径全名	string	64
	调度编号	code	调度编号	string	32
3	变电站 ID	st_id	厂站标识的引用	long	8
4	间隔 ID	bay_id	间隔标识引用	long	8
5	电压类型 ID	bv_id	电压等级标识的引用。	long	8
6	拓扑着色	tpcolor	拓扑值	uchar	1
7	连接点号	nd	连接点号	long	8
8	遥信值	point	网门状态值	uchar	1
9	质量码	status	质量码	int	4
10	责任区	resp_area	责任区值	long	8

A.2.3.8 控制类型参数

表 A.43 主设备遥控（controldigital）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	唯一标识	long	8
4	厂站 ID	st_id	关于厂站表的引用	long	8
5	遥信 ID	pnt_id	遥控的遥信点	long	8
6	遥控类型	ctrl_type	引用“遥控类型”菜单，0 普通遥控，1、直接遥控	uchar	1

7	操作方式	opt_type	操作方式引用“遥控操作类型”菜单，0 单人遥控、1 监护遥控	uchar	1
8	超时时间	timeout	超时时间	int	4

表 A.44 主设备设点（controlsetpnt）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	标识	long	8
2	厂站 ID	st_id	关于厂站表的引用	long	8
3	遥测 ID	alg_id	遥测 ID	long	8
4	控制点号	ctrl_no	控制点号	short	2
5	超时时间	timeout	超时时间	int	4
6	遥控类型	ctrl_type	遥控类型	uchar	1
7	操作方式	opt_type	操作方式引用“遥控操作类型”菜单，0 单人遥控、1 监护遥控	uchar	1

表 A.45 档位升降控制（controltap）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	唯一标识	long	8
2	厂站 ID	st_id	关于厂站表的引用	long	8
3	绕组 ID	trwd_id	调档的变压器绕组 ID	long	8
4	超时时间	timeout	超时时间	int	4
5	操作方式	opt_type	操作方式引用“遥控操作类型”菜单，0 单人遥控、1 监护遥控	uchar	1
6	升档遥控类型	up_pnt_state	升档遥控类型，引用“调档控制类型”，0 普通遥控、1 直接遥控	uchar	1
7	降档控制类型	down_pnt_state	降档控制类型，引用“调档控制类型”，0 普通遥控、1 直接遥控	uchar	1
8	急停控制类型	stop_pnt_state	急停控制类型，引用“调档控制类型”，0 普通遥控、1 直接遥控	uchar	1

表 A.46 辅助设备遥控（auxiliary_controldigital）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	标识	long	8
2	厂站 ID	st_id	关于厂站表的引用	long	8
3	遥信 ID	pnt_id	遥信 ID	long	8
4	控制点号	ctrl_no	控制点号	short	2
5	遥控类型	ctrl_type	遥控类型	uchar	1
6	操作方式	opt_type	操作方式引用“遥控操作类型”菜单，0 单人遥控、1 监护遥控	uchar	1

7	超时时间	timeout	超时时间	int	4

表 A.47 辅助设备设点（auxiliary_controlsetpnt）

序号	属性项	属性项英文名	属性要求	数据类型	数据长度
1	标识	id	标识	long	8
2	厂站 ID	st_id	关于厂站表的引用	long	8
3	遥测 ID	alg_id	遥测 ID	long	8
4	控制点号	ctrl_no	控制点号	short	2
5	超时时间	timeout	超时时间	int	4
6	设点类型	ctrl_type	设点类型	uchar	1
7	操作方式	op_type	操作方式引用“遥控操作类型”菜单，0 单人遥控、1 监护遥控	uchar	1

A.2.3.9 采集信息类

表 A.48 前置主设备遥信信息（fes_yx_define）

序号	属性	属性英文名	属性要求	数据类型	数据长度
1	前置遥信 ID	id	前置遥信 ID	LONG	8
2	厂站 ID	st_id	关于厂站表的引用	LONG	8
3	遥信 ID	yx_id	遥信 ID	LONG	8
4	极性	polarity	极性	UCHAR	1
5	点号	dot_no	点号	INT	4
6	reference 描述	reference	reference 描述	STRING	128

表 A.49 前置主设备遥测信息（fes_yc_define）

序号	属性	属性英文名	属性要求	数据类型	数据长度
1	前置遥测 ID	id	前置遥测 ID	long	8
2	厂站 ID	st_id	关于厂站表的引用	long	8
3	遥测 ID	yc_id	遥测 ID	long	8
4	点号	dot_no	点号	int	4
5	reference 描述	reference	reference 描述	string	128

表 A.50 下行主设备控制信息（send_dc）

序号	域属性	域英文属性	属性要求	数据类型	数据长度
1	前置数据点名	id	前置数据点名	long	8
2	数据点名	psid	数据点名	long	8
3	厂站名	st_id	厂站名	long	8
4	数据点号	index_no	数据点号	short	2
5	检无压点号	novol_index	检无压点号	short	2

6	检同期点号	sync_index	检同期点号	short	2
7	reference 属性	reference	reference 属性	string	128

表 A.51 下行档位遥调信息（send_pd）

序号	域属性	域英文属性	属性要求	数据类型	数据长度
1	前置数据点名	id	前置数据点名	long	8
2	数据点名	psid	数据点名	long	8
3	厂站名	st_id	厂站名	long	8
4	数据点号	index_no	数据点号	short	2
5	升档点号	up_index	升档点号	short	2
6	降档点号	dn_index	降档点号	short	2
7	急停点号	stop_index	急停点号	short	2
8	reference 属性	reference	reference 属性	string	128

表 A.52 下行主设备设点信息（send_sp）

序号	域属性	域英文属性	属性要求	数据类型	数据长度
1	前置数据点名	id	前置数据点名	long	8
2	数据点名	psid	数据点名	long	8
3	厂站名	st_id	厂站名	long	8
4	数据点号	index_no	数据点号	short	2
5	点号	point_no	点号	uchar	1
6	reference 描述	reference	reference 描述	string	128

表 A.53 前置辅设备遥信信息（fes_yx_define_aem）

序号	属性	属性英文名	属性要求	数据类型	数据长度
1	前置遥信 ID	id	前置遥信 ID	long	8
2	厂站 ID	st_id	厂站 ID	long	8
3	遥信 ID	yx_id	遥信 ID	long	8
4	极性	polarity	极性	uchar	1
5	点号	dot_no	点号	int	4
6	reference 描述	reference	reference 描述	string	128

表 A.54 前置辅设备遥测信息（fes_yc_define_aem）

序号	属性	属性英文名	属性要求	数据类型	数据长度
1	前置遥测 ID	id	前置遥测 ID	long	8
2	厂站 ID	st_id	厂站 ID	long	8
3	遥测 ID	yc_id	遥测 ID	long	8
4	点号	dot_no	点号	int	4

5	reference 描述	reference	reference 描述	string	128

表 A.55 下行辅设备控制信息(send_dc_aem)

序号	域属性	域英文属性	属性要求	数据类型	数据长度
1	前置数据点名	id	前置数据点名	long	8
2	数据点名	psid	数据点名	long	8
3	厂站名	st_id	厂站名	long	8
4	数据点号	index_no	数据点号	short	2
5	遥控编号	seq_no	遥控编号	int	4
6	reference 描述	reference	reference 描述	string	128

表 A.56 下行辅设备设点信息(send_sp_aem)

序号	域属性	域英文属性	属性要求	数据类型	数据长度
1	前置数据点名	id	前置数据点名	long	8
2	数据点名	psid	数据点名	long	8
3	厂站名	st_id	厂站名	long	8
4	数据点号	index_no	数据点号	short	2
5	reference 描述	reference	reference 描述	string	128

A.2.3.10 历史告警表

遥信变位数据字典见 A.57。遥测越限数据字典见 A.58。SOE 数据字典见 A.59。

表 A.57 遥信变位（yx_bw）

序号	域英文名	中文描述	数据类型	数据长度
1	occur_time	发生时间	datetime	8
2	restrain_flag	告警抑制标记	int	4
3	content	内容	string	200
4	status	状态	int	4
5	bay_id	间隔 ID	long	8
6	st_id	厂站 ID	long	8
7	resp_area	责任区值	long	8
8	key_id	遥信 ID	long	8
9	milli_second	毫秒数	short	2
10	confirm_node_id	确认节点	long	8
11	confirm_user_id	确认用户	int	4
12	confirm_time	确认时间	datetime	8
13	confirm_status	告警确认状态	int	4
14	customized_group	告警定制分类	int	4

表 A.58 遥测越限（yc_over）

序号	域英文名	中文描述	数据类型	数据长度
1	occur_time	发生时间	datetime	8
2	confirm_node_id	确认节点	long	8
3	confirm_user_id	确认用户	int	4
4	confirm_time	确认时间	datetime	8
5	confirm_status	告警确认状态	int	4
6	customized_group	告警定制分类	int	4
7	restrain_flag	告警抑制标记	int	4
8	content	内容	string	200
9	value	遥测值	float	4
10	status	状态	int	4
11	bay_id	间隔 ID	long	8
12	st_id	厂站 ID	long	8
13	resp_area	责任区值	long	8
14	key_id	遥测 ID	long	8

表 A.59 SOE（yx_soe）

序号	域英文名	中文描述	数据类型	数据长度
1	occur_time	发生时间	datetime	8
2	confirm_node_id	确认节点	long	8
3	confirm_user_id	确认用户	int	4
4	confirm_time	确认时间	datetime	8
5	confirm_status	告警确认状态	int	4
6	customized_group	告警定制分类	int	4
7	content	内容	string	200
8	status	状态	int	4
9	yx_value	遥信值	uchar	1
10	restrain_flag	抑制类型	int	4
11	meas_type	量测类型	int	4
12	resp_area	责任区值	long	8
13	bay_id	间隔 ID	long	8
14	st_id	厂站 ID	long	8
15	key_id	设备名	long	8
16	milli_second	毫秒数	short	2

表 A.60 主设备控制告警（op_ctrl）

序号	域英文名	域中文名	数据类型	数据长度
1	occur_time	发生时间	datetime	8
2	key_id	控制测点	long	8
3	bv_id	电压类型 ID	long	8
4	st_id	厂站 ID	long	8
5	bay_id	间隔 ID	long	8
6	node_id	节点 ID	long	8
7	user_id	用户 ID	long	8
10	resp_area	责任区值	long	8
11	restrain_flag	抑制类型	int	4

12	status	状态	int	4
13	content	内容	string	200
14	customized_group	告警定制分类	int	4
15	confirm_status	告警确认状态	int	4
16	confirm_time	确认时间	datetime	8
17	confirm_user_id	确认用户	int	4
18	confirm_node_id	确认节点	long	8

表 A.61 辅助设备控制告警（op_ctrl_aux）

序号	域英文名	域中文名	数据类型	数据长度
1	occur_time	发生时间	datetime	8
2	key_id	控制测点	long	8
3	bv_id	电压类型 ID	long	8
4	st_id	厂站 ID	long	8
5	bay_id	间隔 ID	long	8
6	node_id	节点 ID	long	8
7	user_id	用户 ID	long	8
10	resp_area	责任区值	long	8
13	meas_value	量测值	float	4
14	status	状态	int	4
15	content	内容	string	200
18	customized_group	告警定制分类	int	4
19	confirm_status	告警确认状态	int	4
20	confirm_time	确认时间	datetime	8
21	confirm_user_id	确认用户	int	4
22	confirm_node_id	确认节点	long	8

表 A.62 二次设备操作告警（relay_op）

序号	域英文名	域中文名	数据类型	数据长度
1	occur_time	发生时间	datetime	8
2	st_id	厂站 ID	long	8
3	dev_id	装置 ID	long	8
4	bay_id	间隔 ID	long	8
5	node_id	节点 ID	long	8
6	user_id	用户 ID	long	8
7	type	操作类型	int	4
8	resp_area	责任区值	long	8

9	status	状态	int	4
10	content	内容	string	200
21	customized_group	告警定制分类	int	4
22	confirm_status	告警确认状态	int	4
23	confirm_time	确认时间	datetime	8
24	confirm_user_id	确认用户	int	4
25	confirm_node_id	确认节点	long	8

A.2.3.11 操作结果表

标志牌操作数据字典见 A.39。人工操作数据字典见 A.40。

表 A.63 人工操作结果（op_info）

序号	域英文名	中文描述	数据类型	数据长度
1	op_id	ID	long	8
2	gua_user_id	监护用户 ID	long	8
3	op_user_id	操作用户 ID	long	8
4	op_node_id	结点 ID	long	8
5	op_time	操作时间	long	8
6	note	备注	string	200
7	data_type	数据类型	int	4
8	dev_type	设备类型	int	4
9	dev_id	设备 ID	long	8
10	bay_id	间隔 ID	long	8
11	st_id	厂站 ID	long	8
12	key_desc	描述	string	200
13	op_type	操作类型	int	4
14	key_id	量测 ID	long	8

表 A.64 主设备限值表（limitsets）

序号	域英文名	域中文名	数据类型	数据长度
1	id	标识	long	8
3	name	中文名称	string	64
4	st_id	厂站 ID	long	8
5	alg_id	遥测 ID	long	8
6	up1_limit	高限 1	float	4
7	low1_limit	低限 1	float	4
8	up2_limit	高限 2	float	4
9	low2_limit	低限 2	float	4
10	limit_status	越限状态	int	4
11	time_point	越限起始时间	int	4
12	yc_value	遥测值	float	4

表 A.65 标志牌操作结果（token_info）

序号	域英文名	中文描述	数据类型	数据长度	备注
1	token_id	标志牌 ID 号	long	8	—
2	st_id	厂站 ID	long	8	—
3	user_name	用户	string	64	—
4	if_display	是否显示	uchar	1	—
5	note	标签	string	200	—
6	token_no	标志牌号	long	8	标志牌定义的引用
7	device_id	设备关键字	long	8	—
8	graph_name	图形名称	string	64	—
9	father_token_id	父牌 ID	long	8	—

表 A.66 对端代结果表（replace_info）

序号	域英文名	域中文名	数据类型	数据长度
1	rped_keyid	被替代遥测 ID	long	8
2	rp_keyid	替代遥测 ID	long	8
3	rp_type	替代类型	uchar	1
4	rped_value	原始值	float	4
5	rped_status	原始状态	int	4
6	rp_factor	替代系数	float	4
7	st_id	厂站 ID	long	8
8	resp_area	责任区值	long	8

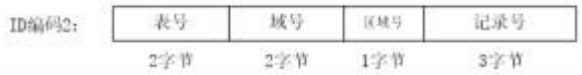
表 A.67 旁路代结果表（bypass_result）

序号	域英文名	域中文名	数据类型	数据长度
1	bypassed_dev	被代设备 Id	long	8
2	bypass_brk	带路开关 Id	long	8
3	bypassed_brk	被代开关 Id	long	8
4	abnormal	异常标志	uchar	1
5	bypassed_p	被代有功量测	long	8
6	bypassed_q	被代无功量测	long	8
7	bypassed_i	被代电流量测	long	8
8	bypass_p	带路有功量测	long	8
9	bypass_q	带路无功量测	long	8
10	bypass_i	带路电流量测	long	8
11	p_value	原始有功值	float	4
12	p_status	原始有功状态	int	4
13	q_value	原始无功值	float	4
14	q_status	原始无功状态	int	4
15	i_value	原始电流值	float	4
16	i_status	原始电流状态	int	4

17	st_id	厂站 ID	long	8

A.2.4 ID 编码

ID 编码唯一标识了一个物理设备、一个逻辑设备或者一个管理信息。在程序内部采用 64 位长整型存储和处理 ID。对 ID 编码进行分段设计和规划，按照表号、域号、区域号、记录号四段。设备 ID 域号为 0，量测的数据 ID 域号为量测类型号。



通过解析 ID 编码获取表号、域号，通过表号在表信息表中获取设备名称；通过表号及域号在域信息表中获取量测类型名称。

编码示例说明如下：

（1）交流线段端点设备 ID 十进制为“116812115351699550”，二进制为“0000000110011111 0000000000000000 00000001 00000000000000000001011110”，按照上述编码规则划分四段后分别为：表号（2 字节）：”0000000110011111”（415），域号（2 字节）：0000000000000000（0），区域号（1 字节）：00000001（1），记录号（3 字节）：00000000000000000001011110（94）。根据表号 415 在表信息表中查询到设备名称为 aclineend，交流线段端点表。

（2）交流线段端点设备有功值遥测的数据 ID 十进制为：“116812244200718430”，转换为二进制后为“0000000110011111 0000000000011110 00000001 00000000000000000001011110”，按照上述编码规则划分四段后分别为：表号（2 字节）：“0000000110011111”（415），域号（2 字节）：0000000000011110（30），区域号（1 字节）：00000001（1），记录号（3 字节）：00000000000000000001011110（94）。根据表号 415，域号 30 在附录 A.域信息表中查询到量测类型名称为 p，有功值。

A.2.5 遥控操作服务号定义

服务号= 4 位功能类型 + 2 位对象类型 + 2 位预留（默认 00）。比如辅设备遥控：10100300 = 1010+03+00。功能类型定义见表 A.68。对象类型见表 A.69。

表 A.68 功能类型定义

功能类型描述	功能类型代码
遥控	1010
设点	1020
调档	1030
顺控	1040
远方操作定值修改	1050
远方操作定值修改执行	1051
批量预置	1060
召唤数据采集	1101
读遥测值	1102
读数据集	1103
召唤状态量	1110
召唤模拟量	1120
远方操作召唤定值	1130
DL/T860 召唤定值	1131
DL/T860 定值修改预置	1132
DL/T860 定值修改执行	1133
召唤定值区号	1140
召唤装置历史信息	1150
召唤通信信息	1160

召唤软压板	1170
RPC 模型调阅	1180
RPC 数据调阅	1190
下发一般文件	2010
召唤一般文件	2020
召唤故障录波文件	2030
调阅文件列表	2110
召唤故障录波文件列表	2120

表 A.69 对象类型

对象类型描述	对象类型代码
无	00
主设备	01
二次设备	02
辅设备	03

A.2.6 平台开发库版本

开发库版本见表 A.70。

表 A.70 开发库版本

库名称	版本号
QT	4.8.4
JDK（安全区 I/II）	1.6
JDK（安全区 IV）	1.8

A.2.7 质量码

遥测、遥信质量码为 int32，采用 bit 位来描述质量为，可同时具备多个质量位。遥测质量码见表 A.71，遥信质量码见表 A.72。

表 A.71 遥测质量码

Bit 位	中文描述	宏定义	备注
3	越合理范围	MENU_STATE_YC_OVERFLOW	—
5	采集中断	MENU_STATE_YC_FAILURE	—
6	不刷新	MENU_STATE_YC_OLDDATA	—
7	可疑	MENU_STATE_YC_SUSPECT	—
8	未初始化	MENU_STATE_YC_UNINIT	—
9	封锁	MENU_STATE_YC_NIS	—
10	告警抑制	MENU_STATE_YC_INHABIT_ALM	—
11	禁止控制	MENU_STATE_YC_INHABIT_CTRL	—
12	控制中	MENU_STATE_YC_CTRL_PENDING	—
13	置数	MENU_STATE_YC_MANUAL	—
14	非实测	MENU_STATE_YC_REAL	—
15	计算	MENU_STATE_YC_CALC	—
17	被对侧代	MENU_STATE_YC_ASIDEBY	—
18	被旁路代	MENU_STATE_YC_BYPASSBY	—
19	异常旁路代	MENU_STATE_YC_ABN_BYPASSBY	—

21	跳变	MENU_STATE_YC_SKIP	—
23	越上限 1	MENU_STATE_YC_HIGHT1	—
24	越下限 1	MENU_STATE_YC_LOW1	—
25	越上限 2	MENU_STATE_YC_HIGHT2	—
26	越下限 2	MENU_STATE_YC_LOW2	—
31	正常	MENU_STATE_YC_NORMAL	—

表 A.72 遥信质量码

Bit 位	中文描述	宏定义	备注
5	采集中断	MENU_STATE_YX_FAILURE	—
8	未初始化	MENU_STATE_YX_UNINIT	—
9	封锁	MENU_STATE_YX_NIS	—
10	告警抑制	MENU_STATE_YX_INHABIT_ALM	—
11	禁止控制	MENU_STATE_YX_INHABIT_CTRL	—
12	控制中	MENU_STATE_YX_CTRL_PENDING	—
13	置数	MENU_STATE_YX_MANUAL	—
14	非实测	MENU_STATE_YX_REAL	—
15	计算	MENU_STATE_YX_CALC	—
19	正常变位	MENU_STATE_YX_STATCHG	人工确认后恢复正常质量码
20	挂牌	MENU_STATE_YX_TAG	—
31	正常	MENU_STATE_YX_NORMAL	—

A.2.8 接口头文件及动态库

A.2.8.1 头文件及动态库说明

A.2.8.1.1 针对附录 A 接口头文件、MCODE 数据结构体头文件以及客户端动态库的存放路径进行规范化管理，存放路径采用统一环境变量$JK_PLATFORM_API 来表示，头文件及动态库名称见表 A.73。

A.2.8.1.2 头文件路径：文件通过#include “mcode/xxx.h”引用头文件。

$JK_PLATFORM_API/jk_api/*.h
$JK_PLATFORM_API/jk_api/mcode/*.h
$JK_PLATFORM_API/jk_api/graph/*.h

A.2.8.1.3 程序编译:

a) Qt 程序编译：.pro 文件中 INCLUDEPATH += $JK_PLATFORM_API/jk_api
b) C/C++动态库、后台服务：makefile 中 g++添加 -I$(JK_PLATFORM_API)/jk_api
c) 动态库编译存放链接路径：$JK_PLATFORM_API/src_lib/*.so
d) 可执行程序编译存放路径：$JK_PLATFORM_API/src_bin/

A.2.8.1.4 程序发布:

a) 动态库发布路径：$JK_PLATFORM_API/lib/*.so
b) 可执行程序发布路径：$JK_PLATFORM_API/bin/

表 A.73 头文件及动态库名称

序号	附录章节	头文件名称	动态库名称	备注
1	A1.1 系统管理服务接口	jk_api/jk_sam_common.h jk_api/jk_proc_inv.h jk_api/jk_sam_service.h	libjk_man.so libjk_sam_common.so libjk_sam_service.so	—
2	A.2 实时数据服务接口	jk_api/jk_rtdb_api.h	libjk_rtdb_client.so	—
3	A.3 历史数据服务接口	jk_api/jk_his_api_curve_param.h jk_api/jk_hisclient.h	libjk_midhc.so	—

4	A.4 商用库服务接口	jk_api/jk_CSqlResultAlignClient.h jk_api/jk_sql_sp_client_base.h	libjk_sql_sp_client.so	—
5	A.5 消息总线接口	jk_api/jk_message_inv.h jk_api/jk_messageheader.h	libjk_rte.so	—
6	A.6 服务总线接口	jk_api/jk_services.h jk_api/jk_remoteCallClient.h	libjk_services.so libjk_remote_call_client.so	—
7	A.7.1 文件服务接口	jk_api/jk_ClientOp.h	libjk_ftp_client.so	—
8	A.7.2 日志服务接口	jk_api/jk_logclient.h	libjk_log_client.so	—
9	A.7.3 安全认证服务接口	jk_api/jk_secbus.h jk_api/jk_secservicebus.h	libjk_secbus.so libseccommon.so libjk_secservicebus.so libsecsoftsm2.so	—
10	A.7.4 文语服务	jk_api/jk_MMIClient.h	libjk_mmi_client.so	—
11	A.8 告警服务	jk_api/jk_IAlarmClientInterface.h jk_api/mcode/jk_warn_confirm_msg_sync_m.h jk_api/mcode/jk_warn_message_new_m.h jk_api/mcode/jk_warn_service_message_m.h	libjk_alarm_client.so	—
12	A.9 权限服务	jk_api/jk_priv_access.h	libjk_priv_client.so	—
13	A.10 模型修改服务	jk_api/jk_model_modify_client.h	libjk_model_modify_client.so	—
14	A.11 操作控制服务接口	jk_api/mcode/jk_seq_ctrl_message_m.h	libjk_m_impl.so	—
15	A.12 实时数据获取接口	jk_api/mcode/jk_change_data_message_m.h	libjk_m_impl.so	—
16	A.13 告警窗右键菜单扩展接口（QT）	jk_api/jk_event_alarm_interfaces.h	—	—
17	A.14 告警窗扩展接口（JAVA）	—	—	—
18	A.15 安全Ⅳ区 JAVA 接口	—	jk-service-api-1.0.jar	—
19	A.16 人机扩展接口（JAVA）	—		—
20	A.17 人机扩展接口（QT）	jk_api/graph/jk_G_CGBuffer.h jk_api/graph/jk_G_CPPInclude.h jk_api/graph/jk_G_DrawObjDefine.h jk_api/graph/jk_G_EnvInfoDefine.h jk_api/graph/jk_G_GlobalDef.h jk_api/graph/jk_G_OpClassDefine.h jk_api/graph/jk_G_OpConstDefine.h jk_api/graph/jk_G_QtInclude.h jk_api/graph/jk_G_SysStruct.h jk_api/graph/interface/jk_G_IApp.h jk_api/graphinterface/jk_G_IAppCallback.h jk_api/graph/interface/jk_G_IAppMessage.h jk_api/graph/interface/jk_G_IMenuPlugin.h	libjk_SCADACallback.so	—
21	A.18 安全Ⅳ区浏览器获取图形接口	无	无	—
22	A.19 自动对点	jk_api/mcode/jk_auto_acceptance_message_m.h	libjk_m_impl.so	—
23	A.20CASE 断面管理	jk_api/jk_mode_manage_struct.h jk_api/jk_IModeManage.h	libjk_case_mode_manage_client.so	—
24	A.21CASE 模型操作	jk_api/jk_model_op_struct.h jk_api/jk_ICaseModelOp.h	libjk_case_model_op_client.so	—
25	A.22 数据采集转发变电站网络安全事件接口	jk_api/mcode/jk_nsnspmsgtype_m.h	libjk_m_impl.so	—

26	A.23 消息通道及类型接口	jk_api/jk_channel_type.h	libjk_channel_type.so	—

A.2.8.2 消息通道及类型说明

消息通道与消息类型采用统一头文件名称、宏定义名称，提供统一的接口根据消息通道及类型名称获取对应的通道和事件类型值,接口定义见附录 A23，应满足以下要求：

a） 消息通道头文件名称：jk_api_msgchan.h；
b） 消息类型头文件名称：jk_api_msgtype.h；
c） 文件存放目录：$JK_PLATFORM_API/msgdef。

A.3 其他说明

A.3.3 M 消息语言说明

A.3.3.1 基本数据类型

支持的基本类型包括：

a) 布尔型 bool
b) 字符型 char
c) 整型 int
d) 长整型 long
e) 无符号字符型 unsigned char
f) 无符号整型 unsigned int
g) 无符号长整型 unsigned long
h) 浮点型 float
i) 双精度浮点 double
j) 结构 struct
k) 字符串 varchar
l) 二进制 binary
m) 向量 vector

A.3.3.2 数据类型定义

A.3.3.2.1 常量的定义

常量定义采用 const 定义方式，常量映射为文件域的 C++常量。
如 const int MAX_LEN=32;

A.3.3.2.2 结构的定义

结构按照 C/C++格式定义结构。不支持递归定义。
不支持如下格式的定义：

```
Typedef struct Test
{
    int x;
};
```

此时应该按如下格式定义：

```
struct Test
{
    int x;
};
```

A.3.3.2.3 向量的定义

直接定义向量即可，例如：
typedef vector<int>vecInt;

若要对 vecInt 类型变量进行序列化，需将其放在结构中。

```
struct Test
{
    vector<int>  vecKey;
}
```

若需要使用 vector<char>表示二进制数据，必须用 binary 代替。

A.3.3.2.4 数组的定义

数组的长度可以为常量数字，也可以为预定义的常量。如：

```
const int MAX_LEN=128;
struct Array
{
    char name[32];
    char addr[MAX_LEN];
};
```

A.3.3.2.5 类型重定义

直接定义即可。例如：

```
typedef vector<int> vecInt;
typedef vector<vecInt>vvInt;
```

使用重定义类型变量实现自动编码、解码时，需要放在结构中。

描述文件要若使用 vector 或 varchar 关键字，须包含 mcode/mtypes.h 头文件。

A.3.3.3 描述文件包含关系

解析器支持文件包含关系，即如果解析文件 A 中包含有另一个文件 B，那么文件 B 中定义的类型在文件 A 中具有作用域，最多支持 10 个包含文件。如下示例：

```
//download.h
#include”public.h”
```

那么 public.h 中定义的类型及常量在 download.h 中具有作用域，可以直接使用。映射后的头文件为：

```
//download_m.h
#include”public_m.h”
```

A.3.3.4 使用方法

以 public.h 为例，使用步骤如下：

a）. 按格式写好结构描述文件 public.h；

b）. 描述文件 public.h 统一放在$SRC/mdf 文件夹中；

c）. 进入应用程序文件夹，如$SRC/sys_admin，输入命令 mcpp $SRC/mdf/public.h，在当前目录生成 public_m.h 和 public_m.cpp 两个文件；

d）. 应用程序中包含 public_m.h 即可，像普通头文件一样使用；

e）. makefile 中要含有$SRC/include 路径（一般标准 makfile 中已经加入），链接库加上 libmstream.so 动态库，即-lmstream；

f）. 无返回值：编码接口为宏 M_CODE，解码接口为 M_DECODE；

g）. 有返回值：编码接口为宏 M_CODE2，解码接口为 M_DECODE2；

h）. 编译

A.3.3.5. 编码、解码接口

A.3.3.5.1 编码、解码接口中含有异常处理代码，因此在编码、解码代码中不会因数据不完整而影响程序的正常运行。

A.3.3.5.2 M_CODE(src_struct, buf_out, buf_size)

参数：

src_struct: 输入参数，待编码结构变量，其类型在*_m.h 定义
buf_out: 输出参数，char*类型指针，指向编码后数据存储空间
buf_size: 输出参数，整数类型，编码数据的长度
buf_out 指向的空间使用后要及时释放，方式为 delete [] buf_out。

A.3.3.5.3 M_DECODE(dest_struct, buf_in, buf_size)

参数：
dest_struct: 输出参数，接收解码结果，其类型在*_m.h 定义
buf_in: 输入参数，char*类型指针，待解码的数据空间
buf_size: 输入参数，整数类型，带解码数据空间的长度

A.3.3.5.4 M_CODE2(src_struct, buf_out, buf_size, rtn)

参数：
前三个参数与 M_CODE 相同，rtn 返回编码结果：rtn ==0 正确编码，<0 编码错误

A.3.3.5.5 M_DECODE2(dest_struct, buf_in, buf_size, rtn)

前三个参数与 M_CODE 相同，rtn 返回解码结果：rtn ==0 解码正确，<0 编码错误

A.4 系统管理服务接口

A.4.1 节点及应用管理接口

A.4.1.1 主机查询接口

应满足以下要求：

a） 接口原型：int RequestService(const int app_id, const int policy, char *host_name, const short context = 0)

b） 接口说明：请求某应用的主机或备机，如果请求任务完成，返回值等于 1，则返回应用运行的服务器；如果请求任务完成，返回值小于 0，则表示没有此应用的主机。

c） 参数列表见表 A.74：

表 A.74 RequestService 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
app_id	const int	In	服务所属应用	是
policy	const int	In	1->查找主机；	是
host_name	char *	Out	返回该应用 所运行的节点名	是
Context	const short	In	应用所属态	是
返回值	Int	Out	=1：查询到相应的应用服务器； <0：失败，没有定位到该应用的主机	—

d） 接口返回值：当某个程序需要当前处于主机应用的机器名称时，可以通过调用 RequestService 接口来得到。其中 app_id 为应用号，policy 一般取值 1，host_name 为返回的主机名称，context 默认是 0，表示取现在所处的态，也可以根据情况取其它值。

e） 接口示例： using namespace JK_API;class CJKServicesManage g_SrvManage;
g_SrvManage.RequestService(AP_PUBLIC, 1, host_name);利用 context 缺省值
g_SrvManage.RequestService(AP_PUBLIC, 1, host_name, context);利用设定好的 context 值

A.4.1.2 备机查询接口

应满足以下要求：

a） 接口原型：int RequestService(const int app_id, const int policy, vector<string> &hosts_name, const short context=0)。

b） 接口说明：请求某应用的主机或备机，如果请求任务完成，返回值等于 1，则返回应用运行

的所有节点名(包括所有主备机)；如果请求任务完成，返回值等于-1，则表示没有此应用的备机服务器。

c） 参数列表见表 A.75：

表 A.75 RequestService 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
app_id	const int	In	服务所属应用	是
policy	const int	In	2->查找备机	是
hosts_name	vector<string> &	Out	返回该应用运行的所有节点名	是
context	const short	In	应用所属态	是
返回值	Int	Out	=1 查询到相应的应用服务器； <0 没有此应用的备机服务器；	—

d） 接口返回值：当某个程序需要当前处于主机应用的机器名称时，可以通过调用 RequestService 接口来得到。其中 app_id 为应用号，policy 一般取值 1，hosts_name 为返回的所有主机名称数组，context 默认是 0，表示取现在所处的态，也可以根据情况取其它值。

e） 接口示例：

```
1) using namespace JK_API;
2) class CJKServicesManage  g_SrvManage；
3) int ret = g_SrvManage.RequestService(AP_PUBLIC, 1, hosts_name);利用 context 缺省值
4) if(ret == 1){}
5) else if(ret == -1){}
6) else{}
7) ret = g_SrvManage.RequestService(AP_PUBLIC, 1, hosts_name, context);利用设定好的
   context 值
```

A.4.1.3 应用运行节点查询接口

应满足以下要求：

a） 函数原型：int GetAppActiveNodes (int app_no, vector<string> &node_name,vector<int> &node_id, const short context = 0)。

b） 功能说明：请求所有活动状态（主机，备机，强制主机，强制备机）的应用节点信息，如果请求任务完成，返回值大于 0，则返回应用运行的服务器节点名集合和服务器节点号集合；如果请求任务完成，返回值小于 0，则表示失败。当某个程序需要当前应用处于所有活动状态机器信息时，可以通过调用 GetAppActiveNodes 接口来得到。其中 app_no 为应用号，node_name 为返回的节点名称集合，node_id 为返回的节点号集合，context 默认是 0，表示取现在所处的态，也可以根据情况取其它值。

c） 参数列表见表 A.76：

表 A.76 GetAppActiveNodes 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
app_no	int	In	服务所属应用	是
node_name	vector<string>&	Out	返回该应用所运行的节点名集合	是
node_id	vector<int>&	Out	返回该应用所运行的节点号集合	是
context	const short	In	应用所属态	否
返回值	Int	Out	>0 查询到相应的应用服务器； <0 失败	—

d） 接口示例：

1) using namespace JK_API;
2) class CJKServicesManage g_SrvManage；
3) g_SrvManage. GetAppActiveNodes (AP_PUBLIC, node_name, node_id);利用 context 缺省值
4) g_SrvManage. GetAppActiveNodes (AP_PUBLIC, node_name, node_id, context);利用设定好的 context 值

A.4.1.4 活动应用查询接口

应满足以下要求：

a） 函数原型：int GetActiveApps (vector<int> &app_id, const short context = 0)。

b） 功能说明：请求本节点上存在的所有活动（主机，备机，强制主机，强制备机，断网，故障）的应用信息，如果请求任务完成，返回值大于 0；如果请求任务完成，返回值小于 0，则表示失败。当某个程序需要当前节点上所有活动状态的应用信息时，可以通过调用 GetActiveApps 接口来得到。其中 app_id 为返回的应用号集合，context 默认是 0，表示取现在所处的态，也可以根据情况取其它值。

c） 参数列表见表 A.77：

表 A.77 GetActiveApps 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
app_id	vector<int>&	Out	应用号集合	是
context	const short	In	应用所属态	否
返回值	Int	Out	>0 查询到相应的应用信息； <0 失败	—

d） 接口示例：

1) using namespace JK_API;
2) class CJKServicesManage g_SrvManage；
3) g_SrvManage. GetActiveApps (node_id);利用 context 缺省值
4) g_SrvManage. GetActiveApps (node_id, context);利用设定好的 context 值

A.4.1.5 查询主备机状态接口

应满足以下要求：

a） 函数原型：int IsOnDuty (const int app_id,const short context = 0)。

b） 功能说明：请求判断应用在本节点上是主机还是备机，如果请求任务完成，返回值等于 1，则表示应用是主机，返回值等于 0 表示应用是备机，返回值等于-1 表示没有该应用；如果是其他小于 0 的返回值，则表示其他系统级错误。当某个程序需要判断某个应用在本节点的主备状态时，可以通过调用 IsOnDuty 接口来得到。其中 app_id 需要查询的应用号，context 默认是 0，表示取现在所处的态，也可以根据情况取其它值。

c） 参数列表见表 A.78：

表 A.78 IsOnDuty 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
app_id	const int	In	应用好	是
Context	const short	In	应用所属态	否
返回值	Int	Out	=1 应用在本节点是主机； =0 应用在本节点是备机； <0 本节点没有该应用主机；	—

d） 接口示例：

using namespace JK_API;

class CJKServicesManage g_SrvManage；
g_SrvManage. IsOnDuty (AP_PUBLIC);利用 context 缺省值。
g_SrvManage. GetActiveApps (AP_PUBLIC, context);利用设定好的 context 值。

A.4.1.6 根据名称获取ID接口

应满足以下要求：

a） 函数原型：int Name_to_ID (const char *app_name, const char *ctx_name,const char *node_name, int &app_id, int &ctx_no, int &node_id)。

b） 功能说明：根据应用名、所属态、节点名，从而获取应用 ID，所属态号、节点 ID，返回值等于 1 表示转换成功，返回值小于 0，表示转换失败。

c） 参数列表见表 A.79：

表 A.79 根据名称获取 Id 接口列表

接口参数	类型	输入/输出	参数说明	是否必填
app_name	const char *	In	应用名	是
ctx_name	const char *	In	应用所属态	是
node_name	const char *	In	节点名	是
app_id	Int &	Out	应用号	是
ctx_no	Int &	Out	应用所属态号	是
node_id	Int &	Out	节点 Id	是
返回值	Int	Out	=1　名称转换 ID 成功 <0　转换失败	—

d） 接口示例：

```
using namespace JK_API;
int main()
{
char app_name[MAX_STRING_LEN] ={0};
char node_name[MAX_STRING_LEN] = {0};
int app_id = 0;
int ctx_no = 0;
int node_id= 0;
strncpy(app_name,"public",MAX_STRING_LEN-1);
strncpy(node_name,"sysadm1",MAX_STRING_LEN-1);
class CJKSamPub sam_common;
ret=sam_common.Name_to_ID(app_name,"",node_name , app_id, ctx_no, node_id);
}
```

A.4.1.7 根据ID获取名称接口

应满足以下要求：

a） 函数原型：int ID_to_Name (const int app_id, const int ctx_no, const int node_id, char *app_name, char *ctx_name, char *node_name)。

b） 功能说明：根据应用 ID，所属态号、节点 ID 获取应用名、所属态、节点名，返回值等于 1 表示转换成功；返回值小于 0，表示转换失败。

c） 参数列表见表 A.80：

表 A.80 根据名称获取 Id 接口列表

接口参数	类型	输入/输出	参数说明	是否必填
app_id	const int	In	应用号	是
ctx_no	const int	In	应用所属态号	是
node_id	const int	In	节点 Id	是
app_name	char *	Out	应用名	是

ctx_name	char *	Out	应用所属态	是
node_name	char *	Out	节点名	是
返回值	Int	Out	=1 ID 转换名称成功 <0 转换失败	—

d） 接口示例：

```
using namespace JK_API;
int main()
{
   char app_name[MAX_STRING_LEN] ={0};
   char ctx_name[MAX_STRING_LEN] ={0};
   char node_name[MAX_STRING_LEN] = {0};
   int app_id = 1600000;
   int ctx_no = 1;
   int node_id= 15;
   class CJKSamPub sam_common;
   ret = sam_common.ID_to_Name(app_id, ctx_no, node_id, app_name, ctx_name, node_name);
}
```

A.4.1.8 获取当前节点ID和IP地址接口

应满足以下要求：

a） 函数原型：int GetLocalNodeInfo (int &local_node_id, string &local_node_ip)。

b） 功能说明：获取本节点 id 和 ip 地址，返回值等于 0 表示获取成功；返回值小于 0，表示获取失败。

c） 参数列表见表 A.81：

表 A.81 获取本节点 id、ip 接口列表

接口参数	类型	输入/输出	参数说明	是否必填
local_node_id	int &	Out	本本节点 id	是
local_node_ip	string &	Out	本节点 ip	是
返回值	Int	Out	=0 获取本节点 id 成功 <0 获取本节点 id 失败	—

d） 接口示例：

```
#include "jk_api/jk_sam_common.h"

using namespace JK_API;
int main(int argc,char *argv[])
{
      int ret = 0;
      class CJKSamPub g_SamCommon;
      int node_id=0;
      string local_node_ip="";
   ret = g_SamCommon.GetLocalNodeInfo(node_id,local_node_ip);
      if(ret < 0)
      {
            printf("GetLocalNodeInfo error!\n");
      }
      else
      {
            printf("GetLocalNodeInfo                                                    success,
MyNodeId:%d,MyNodeIP:%s!\n",node_id,local_node_ip.c_str());
      }
```

```
    return 0;
}
```

A.4.2 进程管理接口

A.4.2.1 基本要求

进程管理为应用程序提供注册进程管理的功能，以便保护应用进程的正常运行。对注册进程进行实时监视和管理，检测到进程掉线后，发送故障告警并尝试恢复。

A.4.2.2 进程注册接口

应满足以下要求：

a） 接口原型：int proc_init(char *context_name, char *app_name, char *proc_name);

b） 接口说明：用于进程注册，入参包括进程所属的态、进程所属的应用、进程注册的名字。应用进程调用此函数时，需要判断函数返回值，注册失败时应退出。可以检查为什么注册名冲突，根据现场环境修改代码逻辑后重新运行和注册。

c） 参数列表见表 A.82：

表 A.82 proc_init 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
context_name	char *	In	态名，进程所属的态	是
app_name	char *	In	应用名，进程所属的应用	是
proc_name	char *	In	进程名，进程注册的名字	是
返回值	Int	Out	>=0 注册成功,数值是进程在共享内存中的注册位置(proc_pos)； <0 注册失败，存在同名进程	—

d） 接口示例：

```
using namespace JK_API;
int main(int argc, char *argv[])
{
    jk_proc_invocation procm;
    int proc_pos = procm.proc_init ("realtime", "public", "xxx_server");

        if (proc_pos < 0 )
        {
            Debug (isDebug, "%s", "Error:procinit register fail\n");
            LogWrite (LOG_ALERT, &log_info, "LINE:%d Error:proc init fail!\n",
                __LINE__);
            exit (0);
        }
    }
```

A.4.2.3 进程取消注册接口

A.4.2.3.1 接口原型：int proc_exit(int proc_pos)。

A.4.2.3.2 接口说明：用户进程向进程管理取消注册。用户进程使用。

A.4.2.3.3 参数列表见表 A.83：

表 A.83 proc_exit 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
proc_pos	Int	In	应用注册进程管理返回的 pos	是
返回值	Int	Out	=1 取消注册成功； <0 取消注册失败	—

A.4.2.3.4 接口示例：

```
using namespace JK_API;
int main(int argc, char *argv[])
{
    jk_proc_invocation procm;
  int proc_pos = 10;
  ret_val = procm.proc_exit(proc_pos);
}
```

A.4.2.3.5 接口原型：int proc_exit(char *context_name, char *app_name, char *proc_name)。

A.4.2.3.6 接口说明：用户进程向进程管理取消注册。用户进程使用。

A.4.2.3.7 参数列表见表 A.84：

表 A.84 proc_exit 参数列表 2

接口参数	类型	输入/输出	参数说明	是否必填
context_name	char *	In	态名，进程所属的态	是
app_name	char *	In	应用名，进程所属的应用	是
proc_name	char *	In	进程名，进程注册的名字	是
返回值	Int	Out	=1 取消注册成功； ＜0 取消注册失败	—

A.4.2.3.8 接口示例：

```
using namespace JK_API;
int main(int argc, char *argv[])
{
    jk_proc_invocation procm;
  ret_val = procm.proc_exit("realtime", "public", "xxx_server");
}
```

A.4.2.4 获取进程运行信息接口

A.4.2.4.1 接口原型：int get_procinfo(int position, JK_PROC_ADM_INFO *p_info)。

A.4.2.4.2 接口说明：获取注册进程的运行状态信息。用户进程使用。

A.4.2.4.3 参数列表见表 A.85：

表 A.85 get_procinfo 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
Position	Int	In	应用注册进程管理的 pos	是
p_info	JK_PROC_ADM_INFO	Out	应用注册进程管理的进程运行信息	是
返回值	Int	Out	=1 获取进程信息成功； ＜0 获取进程信息失败	—

A.4.2.4.4 接口示例：

```
using namespace JK_API;
int main(int argc, char *argv[])
{
    jk_proc_invocation procm;
  int proc_pos = 10;
  JK_PROC_ADM_INFO p_info;
  ret_val = procm.get_procinfo(proc_pos，&p_info);
}
```

A.4.2.4.5 接口原型：int get_procinfo(char *context_name, char *app_name, char *proc_name, JK_PROC_ADM_INFO *p_info)。

A.4.2.4.6 接口说明：获取注册进程的运行状态信息。用户进程使用。

A.4.2.4.7　参数列表见表 A.86：

表 A.86 get_procinfo 参数列表 2

接口参数	类型	输入/输出	参数说明	是否必填
context_name	char *	In	态名，进程所属的态	是
app_name	char *	In	应用名，进程所属的应用	是
proc_name	char *	In	进程名，进程注册的名字	是
p_info	JK_PROC_ADM_INFO	Out	应用注册进程管理的进程运行信息	是
返回值	Int	Out	=1　获取进程信息成功； ＜0　获取进程信息失败	是

A.4.2.4.8　接口示例：

```
using namespace JK_API;
int main(int argc, char *argv[])
{
    jk_proc_invocation procm;
    JK_PROC_ADM_INFO  p_info;
    ret_val = procm.get_procinfo("realtime", "public", "xxx_server"，&p_info);
}
```

A.4.2.5　获取进程运行状态接口

A.4.2.5.1　接口原型：int is_proc_run(char *context_name, char *app_name, char *proc_name)。

A.4.2.5.2　接口说明：获取进程的运行状态信息。用户进程使用。

A.4.2.5.3　参数列表见表 A.87：

表 A.87 is_proc_run 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
context_name	char *	In	进程所属态	是
app_name	char *	In	进程所属应用	是
proc_name	char *	In	进程注册名	是
返回值	Int	In	=1　进程在线； ＜0　进程不在线	—

A.4.2.5.4　接口示例：

```
using namespace JK_API;
int main(int argc, char *argv[])
{
    int proc_status=0;
    jk_proc_invocation procm;
    char  context_name[MAX_STRING_LEN]=”realtime”;
    char  app_name[MAX_STRING_LEN]=”public”;
  char  proc_name[MAX_STRING_LEN]=”rtdb_server_SCADA”;
    proc_status= procm .is_proc_run (context_name, app_name, proc_name) ;
}
```

A.5　实时数据服务接口

A.5.1　打开表

A.5.1.1　接口原型：int Open(const int app_no, const int table_no, const short context_no = 0)。

A.5.1.2　接口说明：该接口在使用实时库的接口对某张表进行操作之前必须先打开这张表，用于加载访问环境，如果打开成功，则结果返回 0；如果表打开失败结果返回小于 0。网络接口只校验参数

合法性，不校验输入应用号、表号、态号是否存在。

A.5.1.3　参数列表见表 A.88：

表 A.88 Open 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
app_no	const int	In	所访问实时库从属应用的应用号	是
table_no	const int	In	所访问实时库表号，根据模型数据字典按需获取。	是
context_no	const short	In	所访问实时库从属态的态号	是
返回值	int	Out	=0　成功 <0　失败	—

A.5.1.4　接口示例（CJKTableOp 本地接口访问）：

```
int main()
{
    int app_no = 100000;
    int context_no = 1;  //实时态
    int table_no = 220;  //模型数据表索引号
    CJKTableOp table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
        //环境加载失败
        //错误处理
    }
    else
    {
        //环境加载成功
        //后续处理
    }
}
```

A.5.1.5　接口示例（CJKTableNet 网络接口访问）：

```
int main()
{
   int app_no = 100000;
    int context_no = 1;  //实时态
    int table_no = 220;  //模型数据表索引号
    CJKTableNet table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
        //环境加载失败
        //错误处理
    }
    else
    {
        //环境加载成功
        //后续处理
    }
}
```

A.5.2　获取表内容

A.5.2.1 接口原型：int TableGet(const char* field_name, char** buffer_ptr, int& buffer_size)

A.5.2.2 接口说明：该接口用于根据字段取本张表所有记录的某个域，多个字段名称用逗号隔开，如果获取记录成功，则结果返回大于等于 0；如果表获取记录失败结果返回小于 0。

A.5.2.3 参数列表见表 A.89：

表 A.89 TableGet 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
field_name	const char*	In	访问的字段属性名称	是
buffer_ptr	char**	Out	输出结果内存地址，内存地址由接口分配，又调用者通过 free 释放	是
buffer_size	int&	Out	输出结果内存大小	是
返回值	Int	out	>=0 成功 <0 失败	—

A.5.2.4 接口示例（CJKTableOp 本地接口访问）：

```
int main()
{
   int app_no = 100000;
      int context_no = 1; //实时态
      int table_no = 220; //模型数据表索引号
      CJKTableOp table_operation_obj;
      int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
      if(ret_code <0)
      {
            return ret_code;
      }
      char* field_name = "id,name,i,i_qual,p,p_qual";//获取的表域属性
//定义获取数据的结构体，结构顺序、数据类型与上述表域一致
struct RESULT_CONTENT
{
      long id;
      char name[64];
      float i;
      int i_qual;
      float p;
      int p_qual;
};
char* get_data_ptr = NULL;
int get_data_size = 0;
ret_code = table_operation_obj.TableGet(field_name, &get_data_ptr, get_data_size);//获取表全数据
if(ret_code < 0)
{
      //访问失败
}
else
{
      //访问成功，输出结果
struct RESULT_CONTENT* p_content = (struct RESULT_CONTENT*)get_data_ptr;
            printf("1st record:\n\tid:\b%ld\n\tname:\b%s\n\ti:\b%f\n\ti_qual:\b%d\n",
                  p_content->id, p_content->name, p_content->i, p_content->i_qual);
}
      return 0;
}
```

A.5.2.5　接口示例（CJKTableNet 网络接口访问）：

```
int main()
{
int app_no = 100000;
    int context_no = 1;  //实时态
    int table_no = 220;  //模型数据表索引号
    CJKTableNet table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
        return ret_code;
    }
    char* field_name = "id,name,i,i_qual,p,p_qual";
struct RESULT_CONTENT
{
    long id;
    char name[64];
    float i;
    int i_qual;
    float p;
    int p_qual;
};
char* get_data_ptr = NULL;
int get_data_size = 0;
ret_code = table_operation_obj.TableGet(field_name, &get_data_ptr, get_data_size);
if(ret_code < 0)
{
    //访问失败
}
else
{
    //访问成功，输出结果
struct RESULT_CONTENT* p_content = (struct RESULT_CONTENT*)get_data_ptr;
        printf("1st record:\n\tid:\b%ld\n\tname:\b%s\n\ti:\b%f\n\ti_qual:\b%d\n",
            p_content->id, p_content->name, p_content->i, p_content->i_qual);
}
    return 0;
}
```

A.5.3　按关键字获取表内容

A.5.3.1　接口原型：int TableGetByKey(const char* key_ptr, const char* field_name,, char* buffer_ptr, constint buffer_size)

A.5.3.2　接口说明：该接口用于取出某张表关键字为 key_ptr 的记录，如果获取记录成功，则结果返回大于等于 0；如果表获取记录失败结果返回小于 0。

A.5.3.3　参数列表见表 A.90：

表 A.90 TableGetByKey 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
key_ptr	const char*	In	关键字	是
field_name	const char*	In	访问的字段名称	是
buffer_ptr	char*	Out	存放取出数据的指针，需要调用者分配内存	是
buffer_size	const int	In	参数 buf_ptr 分配内容的大小	是
返回值	int	Out	>=0 成功 <0 失败	—

A.5.3.4 接口示例（CJKTableOp 本地接口访问）：

```
int main()
{
int app_no = 100000;
    int context_no = 1;  //实时态
    int table_no = 220;
    CJKTableOp table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
        return ret_code;
    }

    char* field_name = "id,name,i,i_qual,p,p_qual";

struct RESULT_CONTENT
{
    char name[64];
    float i;
    int i_qual;
    float p;
    int p_qual;
};
    struct RESULT_CONTENT result;
    long id = 112347912548871;
    ret_code = table_operation_obj.TableGetByKey((char*)&id, field_name, (char*)&result,
sizeof(struct RESULT_CONTENT));
    if(ret_code < 0)
    {
        //访问失败
    }
    else
    {
    //访问成功，输出结果
        printf("record:\n\tid:\b%ld\n\tname:\b%s\n\ti:\b%f\n\ti_qual:\b%d\n",
            id, result.name, result.i, result.i_qual);
    }
    return 0;
}
```

A.5.3.5 接口示例（CJKTableNet 网络接口访问）：

```
int main()
{
    int app_no = 100000;
```

```
        int context_no = 1;  //实时态
        int table_no = 220;
        CJKTableNet table_operation_obj;
        int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
        if(ret_code <0)
        {
            return ret_code;
        }

        char* field_name = "id,name,i,i_qual,p,p_qual";

    struct RESULT_CONTENT
    {
        char name[64];
        float i;
        int i_qual;
        float p;
        int p_qual;
    };
        struct RESULT_CONTENT result;
        long id = 112347912548871;
        ret_code = table_operation_obj.TableGetByKey((char*)&id, field_name, (char*)&result,
sizeof(struct RESULT_CONTENT));
        if(ret_code < 0)
        {
            //访问失败
        }
        else
        {
        //访问成功，输出结果
            printf("record:\n\tid:\b%ld\n\tname:\b%s\n\ti:\b%f\n\ti_qual:\b%d\n",
                .id, result.name, result.i, result.i_qual);
        }
        return 0;
    }
```

A.5.4 写入表内容

A.5.4.1 接口原型：int TableWrite(const char* buffer_ptr, const int buffer_size)。

A.5.4.2 接口说明：该接口用于写入多条记录，写入的必须是表中所不存在的记录，如果写入成功，则结果返回大于等于 0；如果写入失败结果返回小于 0。写入前需要判断表中是否有重复的 记录，如果有需要先调用 DeleteRecord 后再写入，否则写入失败。

A.5.4.3 参数列表见表 A.91：

表 A.91 TableWrite 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
buffer_ptr	const char*	In	写入的数据	是
buffer_size	const int	In	写入数据的大小	是
返回值	Int	Out	>0 成功（该返回值表示写入成功的记录条数） <=0 失败	—

A.5.4.4 接口示例（CJKTableOp 本地接口访问）：

```
    int main()
```

```
{
    int app_no = 100000;
    int context_no = 1;  //实时态
    int table_no = 220;
    CJKTableOp table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
        return ret_code;
    }

struct RECORD_CONTENT
{
    long id;
    char name[64];
    float i;
    int i_qual;
    float p;
    int p_qual;
};

    struct RECORD_CONTENT one_record;
    one_record.id = 112347912548871; //必须填写关键字
    strncpy(one_record.name, "test_record1",63);
    one_record.name[63]=0;

    ret_code = table_operation_obj.TableWrite((char*)&one_record, sizeof(struct
RECORD_CONTENT));
    if(ret_code <= 0)
    {
        //写入失败
    }
    else
    {
        //写入成功
    }

    return 0;
}
```

A.5.4.5 接口示例（CJKTableNet 网络接口访问）：

```
int main()
{
int app_no = 100000;
    int context_no = 1;  //实时态
    int table_no = 220;
    CJKTableNet table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
        return ret_code;
    }

struct RECORD_CONTENT
```

```
{
    long id;
    char name[64];
    float i;
    int i_qual;
    float p;
    int p_qual;
};

    struct RECORD_CONTENT one_record;
    one_record.id = 112347912548871; //必须填写关键字
    strncpy(one_record.name, "test_record1",63);
    one_record.name[63]=0;

    ret_code = table_operation_obj.TableWrite((char*)&one_record, sizeof(struct
RECORD_CONTENT));
    if(ret_code <= 0)
    {
        //写入失败
    }
    else
    {
        //写入成功
    }

    return 0;
}
```

A.5.5 修改表内容

A.5.5.1 接口原型：int TableModifyByKey(const char* key_ptr, const char* field_name, const char* buffer_ptr, const int buffer_size)。

A.5.5.2 接口说明：该接口用于按域号修改某张表的关键字为 key_ptr 的记录的一个域，如果修改成功，则结果返回大于等于 0；如果修改失败结果返回小于 0。如果调用网络接口返回更新成功的 条数。

A.5.5.3 参数列表见表 A.92：

表 A.92 TableModifyByKey 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
key_ptr	const char*	In	关键字	是
field_no	const char*	In	域号	是
buffer_ptr	const char*	In	修改数据的指针	是
buffer_size	const int	In	修改数据的大小	是
返回值	Int	Out	>=0 成功 <0 失败	—

A.5.5.4 接口示例（CJKTableOp 本地接口访问）：

```
int main()
{
int app_no = 100000;
    int context_no = 1;  //实时态
    int table_no = 220;
    CJKTableOp table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
```

```
        return ret_code;
    }

    char* field_name = "i";
  float i_value = 100.12;
    long id = 112347912548871;
    ret_code = table_operation_obj.TableModifyByKey((char*)&id, field_name, (char*)&i_value,
sizeof(float));
    if(ret_code < 0)
    {
        //修改失败
    }
    else
    {
    //修改成功
    }
    return 0;
}
```

A.5.5.5 接口示例（CJKTableNet 网络接口访问）：

```
int main()
{
    int app_no = 1000; //1000 SCADA 应用，1600000 public 应用，400000 fes 应用
    int context_no = 1; //实时态
    int table_no = 220;
    CJKTableNet table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
        return ret_code;
    }

    char* field_name = "i";
  float i_value = 100.12;
    long id = 112347912548871;
    ret_code = table_operation_obj.TableModifyByKey((char*)&id, field_name, (char*)&i_value,
sizeof(float));
    if(ret_code < 0)
    {
        //修改失败
    }
    else
    {
    //修改成功
    }
    return 0;
}
```

A.5.6 更新表内容

A.5.6.1 接口原型：int TableUpdate(const char* buf_ptr, const int buf_size)。

A.5.6.2 接口说明：该接口用于更新表内的记录，在表内如果有这条记录就修改，没有就写入，如果更新成功，则结果返回大于等于 0；如果更新失败结果返回小于 0，使用方法与 TableModify 相同，只不过 TableModify 只修改存在的记录，并不写入新的记录。

A.5.6.3 参数列表见表 A.93：

表 A.93 DeleteRecord 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
buf_ptr	char*	In	更新的数据	是
buf_size	int	In	更新数据的大小	
返回值	Int	Out	>=0 成功 <0 失败	—

A.5.6.4 接口示例（CJKTableOp 本地接口访问）：

```
int main()
{
int app_no = 100000;
    int context_no = 1;  //实时态
    int table_no = 220;
    CJKTableOp table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
        return ret_code;
    }

struct RECORD_CONTENT
{
    long id;
    char name[64];
    float i;
    int i_qual;
    float p;
    int p_qual;
};

    struct RECORD_CONTENT one_record;
    one_record.id = 112347912548871; //必须填写关键字
    strncpy(one_record.name, "test_record1",63);
    one_record.name[63]=0;

    ret_code = table_operation_obj.TableUpdate((char*)&one_record, sizeof(struct
RECORD_CONTENT));
    if(ret_code < 0)
    {
        //更新失败
    }
    else
    {
        //更新成功
    }

    return 0;
}
```

A.5.6.5 接口示例（CJKTableNet 网络接口访问）：

```
int main()
{
int app_no = 100000;
    int context_no = 1;  //实时态
    int table_no = 220;
```

```
CJKTableNet table_operation_obj;
int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
if(ret_code <0)
{
    return ret_code;
}
struct RECORD_CONTENT
{
    long id;
    char name[64];
    float i;
    int i_qual;
    float p;
    int p_qual;
};

    struct RECORD_CONTENT one_record;
    one_record.id = 11234791254887l; //必须填写关键字
    strncpy(one_record.name, "test_record1",63);
    one_record.name[63]=0;

    ret_code = table_operation_obj.TableUpdate((char*)&one_record, sizeof(struct RECORD_CONTENT));
    if(ret_code < 0)
    {
        //更新失败
    }
    else
    {
        //更新成功
    }

    return 0;
}
```

A.5.7 删除表内容

A.5.7.1 接口原型：int DeleteRecord(const char* key_ptr)。

A.5.7.2 接口说明：该接口用于根据关键字删除一条记录或几条记录，如果删除成功，则结果返回大于等于 0；如果删除失败结果返回小于 0。

A.5.7.3 参数列表见表 A.94：

表 A.94 DeleteRecord 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
key_ptr	const char*	In	要删除的记录的关键字	是
返回值	Int	Out	>=0 成功 <0 失败	—

A.5.7.4 接口示例（CJKTableOp 本地接口访问）：

```
int main()
{
    int app_no = 100000;
    int context_no = 1;  //实时态
    int table_no = 220;
    CJKTableOp table_operation_obj;
```

```
        int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
        if(ret_code <0)
        {
            return ret_code;
        }
        long id = 112347912548871;
        ret_code = table_operation_obj. DeleteRecord ((char*)&id);
        if(ret_code < 0)
        {
            //删除失败
        }
        else
        {
       //删除成功
        }
        return 0;
    }
```

A.5.7.5 接口示例（CJKTableNet 网络接口访问）：

```
    int main()
    {
    int app_no = 100000;
        int context_no = 1;  //实时态
        int table_no = 220;
        CJKTableNet table_operation_obj;
        int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
        if(ret_code <0)
        {
            return ret_code;
        }
        long id = 112347912548871;
        ret_code = table_operation_obj. DeleteRecord ((char*)&id);
        if(ret_code < 0)
        {
            //删除失败
        }
        else
        {
       //删除成功
        }
        return 0;
    }
```

A.5.8 条件查询表数据

A.5.8.1 本地接口原型：int ConGet(const int get_field_no, const int con_field_no, const char* con_field_value, char** buf_ptr, int& buf_size)。

A.5.8.2 接口说明：该接口用于根据传入的域号查询指定的域号的值，如果查询成功，则结果返回大于等于 0；如果查询失败结果返回小于 0。

A.5.8.3 参数列表见表 A.95：

表 A.95 ConGet 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
get_field_no	int	In	查询域号	是
con_field_no	int	In	条件域号	是
con_field_value	char**	Out	条件域值	
buf_ptr	char**	In	取出数据的二重指针	是
buf_size	int&	In	输出结果内存大小	是
返回值	Int	Out	>=0 成功 <0 失败	—

A.5.8.4 接口示例（CJKTableOp 本地接口访问）：

```
int main()
{
int app_no = 100000;
    int context_no = 1;  //实时态
    int table_no = 220;
    CJKTableOp table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
        return ret_code;
    }
struct RESULT_CONTENT
{
    long id;
    char name[64];
    float i;
    int i_qual;
    float p;
    int p_qual;
};
    char* buf_ptr = NULL;
    int buf_size = 0;
long alg_id = 115404869971476481;
ret_code = table_operation_obj.ConGet(1, 5, (char*)&alg_id, &buf_ptr, buf_size);
//成功则返回成功查询的记录条数，否则返回小于 0 的数
if(ret_code < 0)
    {
        //访问失败
    }
    else
    {
        //访问成功，输出结果
    }

free(buf_ptr);
    return 0;
}
```

A.5.8.5 网络接口原型：int ConGet (const int get_field_no, const int con_field_no,const char* con_field_value, const int con_value_size, CJKBuffer& buf_base)。

A.5.8.6 接口说明：该接口用于根据传入的域号查询指定的域号的值，如果查询成功，则结果返回大于等于 0；如果查询失败结果返回小于 0。

A.5.8.7 参数列表见表 A.96：

表 A.96 ConGet 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
get_field_no	const int	In	查询域号	是
con_field_no	const int	In	条件域号	是
con_field_value	const char**	In	条件域值指针	是
con_value_size	const int	In	条件域值长度	是

buf_base	CJKBuffer&	Out	输出结果内存结构 CJKBuffer，
			CJKBuffer 常用方法：
			GetBufPtr() //获取内存首地址
			GetLength() //获取内存长度
				是

返回值	Int	Out	>=0 成功 <0 失败	—

A.5.8.8 接口示例（CJKTableNet 网络接口访问）：

```
int main()
{
    int app_no = 100000;
    int context_no = 1;  //实时态
    int table_no = 220;
    CJKTableNet table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
        return ret_code;
    }
struct RESULT_CONTENT
{
    long id;
    char name[64];
    float i;
    int i_qual;
    float p;
    int p_qual;
};

CJKBuffer buff;
long alg_id = 115404869971476481;
ret_code = table_operation_obj.ConGet(1, 5, (char*)&alg_id, sizeof(long),  buff);
//成功则返回成功查询的记录条数，否则返回小于 0 的数
if(ret_code < 0)
    {
        //访问失败
    }
    else
    {
        //访问成功，输出结果
    }

    return 0;
}
```

A.5.9　SQL获取表内容

A.5.9.1 接口原型：int SqlGet(const char* str_sql, char** buffer_ptr, int& buffer_size)。

A.5.9.12 接口说明：该接口用于根据传入的 sql 语句返回 CJKBuffer 查询结果，如果查询成功，则结

果返回大于等于 0；如果查询失败结果返回小于 0。

A.5.9.3 参数列表见表 A.97：

表 A.97 SqlGet 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
str_sql	const char*	In	Sql 语句	是
buffer_ptr	char**	Out	输出结果内存地址，内存地址由接口分配，又调用者通过 free 释放	是
buffer_size	int&	Out	输出结果内存大小	是
返回值	Int	Out	>=0 成功 <0 失败	—

A.5.9.4 接口示例（CJKTableOp 本地接口访问）：

```
int main()
{
int app_no = 100000;
    int context_no = 1;  //实时态
    int table_no = 220;
    CJKTableOp table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
        return ret_code;
    }

    char* sql_statmemt = "select id,name,i,i_qual,p,p_qual from test_table where name like '%厂站1%'";

struct RESULT_CONTENT
{
    long id;
    char name[64];
    float i;
    int i_qual;
    float p;
    int p_qual;
};
    char* get_data_ptr = NULL;
    int get_data_size = 0;

    ret_code = table_operation_obj.SqlGet(sql_statmemt, &get_data_ptr, get_data_size);
    if(ret_code < 0)
    {
        //访问失败
    }
    else
    {
        //访问成功，输出结果
        struct RESULT_CONTENT* p_content = (struct RESULT_CONTENT*)get_data_ptr;
        printf("1st record:\n\tid:\b%ld\n\tname:\b%s\n\ti:\b%f\n\ti_qual:\b%d\n",
            p_content->id, p_content->name, p_content->i, p_content->i_qual);
    }

    return 0;
```

```
}
```

A.5.9.5 接口示例（CJKTableNet 网络接口访问）：

```
int main()
{
    int app_no = 100000;
    int context_no = 1;  //实时态
    int table_no = 220;
    CJKTableNet table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
        return ret_code;
    }

    char* sql_statmemt = "select id,name,i,i_qual,p,p_qual from test_table where name like '%厂站1%'";

struct RESULT_CONTENT
{
    long id;
    char name[64];
    float i;
    int i_qual;
    float p;
    int p_qual;
};
    char* get_data_ptr = NULL;
    int get_data_size = 0;

    ret_code = table_operation_obj.SqlGet(sql_statmemt, &get_data_ptr, get_data_size);
    if(ret_code < 0)
    {
        //访问失败
    }
    else
    {
        //访问成功，输出结果
        struct RESULT_CONTENT* p_content = (struct RESULT_CONTENT*)get_data_ptr;
        printf("1st record:\n\tid:\b%ld\n\tname:\b%s\n\ti:\b%f\n\ti_qual:\b%d\n",
            p_content->id, p_content->name, p_content->i, p_content->i_qual);
    }

    return 0;
}
```

A.5.10 获取表属性信息

A.5.10.1 接口原型：int GetFieldPara(vector<struct FIELD_BASE_INFO_JK>& field_info_vec)

A.5.10.2 参数列表见表 A.98：

表 A.98 GetFieldPara 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
field_info_vec	vector<struct FIELD_BASE_INFO_JK>	Out	属性信息	是
返回值	Int	Out	>=0 成功 <0 失败	—

A.5.10.3 参数说明：

```
struct FIELD_BASE_INFO_JK
{
    int          offset;//属性偏移
    int          field_length;//属性长度
    short        field_no;//属性号
    unsigned char   data_type;//属性类型
    unsigned char   is_keyword;//是否关键字
};
```

A.5.10.4 接口示例：

（CJKTableOp 本地接口访问）：

```
int main()
{
    int app_no = 100000;
    int context_no = 1;  //实时态
    int table_no = 220;
    CJKTableOp table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
        return ret_code;
    }
    std::vector<struct FIELD_BASE_INFO_JK>  field_info_vec;
    ret_code = table_operation_obj.GetFieldPara(field_info_vec);
    if(ret_code <0)
    {
        return ret_code;
    }
    else
    {
      //打印域信息
    }
    return 0;
}
```

（CJKTableNet 网络接口访问）：

```
int main()
{
    int app_no = 100000;
    int context_no = 1;  //实时态
    int table_no = 220;
    CJKTableNet table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
```

```
        return ret_code;
    }
    std::vector<struct FIELD_BASE_INFO_JK>  field_info_vec;
    ret_code = table_operation_obj.GetFieldPara(field_info_vec);
    if(ret_code <0)
    {
        return ret_code;
    }
    else
    {
      //打印域信息
    }
    return 0;
}
```

A.5.11 根据表名获取表号

A.5.11.1 接口原型：int GetTableNoByName(int& table_no, const char* table_name, const bool is_eng = true)

A.5.11.2 参数列表见表 A.99：

表 A.99 GetTableNoByName 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
table_no	int&	Out	表号	
table_name	const char*	In	表名	是
is_eng	const bool	In	是否英文，缺省为是	
返回值	Int	Out	>=0 成功 <0 失败	—

A.5.11.3 接口示例：

（CJKTableOp 本地接口访问）：

```
int main()
{
    int tab_no = 0;
    char* table_name = "sys_table_info";
    CJKTableOp table_operation_obj;
    int ret_code = table_operation_obj. GetTableNoByName (tab_no, table_name);
    if(ret_code <0)
    {
        return ret_code;
    }
    else
    {
      //打印表号
    }
    return 0;
}
```

（CJKTableNet 网络接口访问）：

```
int main()
{
    int tab_no = 0;
    char* table_name = "sys_table_info ";
    CJKTableNet table_operation_obj;
```

```
    int ret_code = table_operation_obj. GetTableNoByName (tab_no, table_name);
    if(ret_code <0)
    {
        return ret_code;
    }
    else
    {
      //打印表号
    }
    return 0;
}
```

A.5.12 根据表号获取表名

A.5.12.1 接口原型：int GetTableNameByNo (char* table_name, const int table_no, const bool is_eng = true)

A.5.12.2 参数列表见表 A.100：

表 A.100 GetTableNameByNo 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
table_name	char*	Out	表名	
table_no	const int	In	表号	是
is_eng	const bool	In	是否英文，缺省为是	
返回值	Int	Out	=0 成功 <0 失败	—

A.5.12.3 接口示例：

（CJKTableOp 本地接口访问）：

```
int main()
{
    int tab_no = 1;
    char table_name[64] = {0};
    CJKTableOp table_operation_obj;
    int  ret_code = table_operation_obj. GetTableNameByNo (table_name , tab_no);
  if(ret_code != 0)
    {
        return ret_code;
    }
    else
    {
      //打印表名
    }
    return 0;
}
```

（CJKTableNet 网络接口访问）：

```
int main()
{
    int tab_no = 1;
    char table_name[64] = {0};
  CJKTableNet table_operation_obj;
    int  ret_code = table_operation_obj. GetTableNameByNo (table_name , tab_no);
```

```
    if(ret_code != 0)
      {
          return ret_code;
      }
      else
      {
        //打印表号
      }
      return 0;
}
```

A.5.13 根据域名获取域号

A.5.13.1 接口原型：int GetFieldNoByName (int& field_no, const char* field_name, const bool is_eng = true)

A.5.13.2 参数列表见表 A.101：

表 A.101 GetFieldNoByName 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
field _no	int&	Out	域号	—
field _name	const char*	In	域名	是
is_eng	const bool	In	是否英文，缺省为是	—
返回值	Int	Out	=0 成功 <0 失败	—

A.5.13.3 接口示例：

（CJKTableOp 本地接口访问）：

```
int main()
{
    int app_no = 1600000;
    int context_no = 1; //实时态
    int table_no = 1;
    CJKTableOp table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
        return ret_code;
    }

  int field_no = 0;
    char* field_name = "table_name_eng";
    ret_code = table_operation_obj. GetFieldNoByName (field_no, field_name);
  if(ret_code != 0)
    {
        return ret_code;
    }
    else
    {
      //打印域号
    }
    return 0;
}
```

（CJKTableNet 网络接口访问）：

```
int main()
{
    int app_no = 1600000;
    int context_no = 1;  //实时态
    int table_no = 1;
    CJKTableNet table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
        return ret_code;
    }

  int field_no = 0;
    char* field_name = "table_name_eng";
    ret_code = table_operation_obj. GetFieldNoByName (field_no, field_name);
  if(ret_code != 0)
    {
        return ret_code;
    }
    else
    {
      //打印域号
    }
    return 0;
}
```

A.5.14 根据批量域英文名获取域号

A.5.14.1 接口原型：int GetFieldNoByName (std::vector<int>& vec_field_no, const char* field_name)

A.5.14.2 参数列表见表 A.102：

表 A.102 GetFieldNoByName 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
vec_field_no	std::vector<int>&	Out	域号 vector	—
field _name	const char*	In	域名，用“,”隔开	是
返回值	Int	Out	=0 成功 <0 失败	—

A.5.14.3 接口示例：

（CJKTableOp 本地接口访问）：

```
int main()
{
    int app_no = 1600000;
    int context_no = 1;  //实时态
    int table_no = 1;
    CJKTableOp table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
        return ret_code;
    }
```

```
  std::vector<int> vec_field_no;
    char* field_name = "table_id, table_name_eng, table_name_chn";
    ret_code = table_operation_obj. GetFieldNoByName (vec_field_no, field_name);
  if(ret_code != 0)
    {
        return ret_code;
    }
    else
    {
      //批量打印域号
    }
    return 0;
}
```

（CJKTableNet 网络接口访问）：

```
int main()
{
    int app_no = 1600000;
    int context_no = 1;  //实时态
    int table_no = 1;
    CJKTableNet table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
        return ret_code;
    }

  std::vector<int> vec_field_no;
    char* field_name = "table_id, table_name_eng, table_name_chn";
    ret_code = table_operation_obj. GetFieldNoByName (vec_field_no, field_name);
  if(ret_code != 0)
    {
        return ret_code;
    }
    else
    {
      //批量打印域号
    }
    return 0;
}
```

A.5.15 根据域号获取域名

A.5.15.1　接口原型：int　GetFieldNameByNo (char* field_name, const int field_no, const bool is_eng = true)

A.5.15.2 参数列表见表 A.103：

表 A.103 GetFieldNameByNo 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
field_name	char*	Out	域名	—
field _no	const int	In	域号	是

is_eng	const bool	In	是否英文，缺省为是	—
返回值	Int	Out	=0 成功 <0 失败	—

A.5.15.3 接口示例：

（CJKTableOp 本地接口访问）：

```
int main()
{
    int app_no = 1600000;
    int context_no = 1;  //实时态
    int table_no = 1;
    CJKTableOp table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
        return ret_code;
    }

  int field_no = 1;
    char field_name[64] = {0};
    ret_code = table_operation_obj. GetFieldNameByNo (field_name , field_no);
    if(ret_code != 0)
    {
        return ret_code;
    }
    else
    {
      //打印域名
    }
    return 0;
}
```

（CJKTableNet 网络接口访问）：

```
int main()
{
    int app_no = 1600000;
    int context_no = 1;  //实时态
    int table_no = 1;
    CJKTableNet table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
        return ret_code;
    }

  int field_no = 1;
    char field_name[64] = {0};
    ret_code = table_operation_obj. GetFieldNameByNo (field_name , field_no);
  if(ret_code != 0)
    {
        return ret_code;
    }
    else
```

```
    {
      //打印域名
    }
    return 0;
}
```

A.5.16 根据多条记录的关键字查询多个域的域值

A.5.16.1 接口原型：int TableGetByKey(const char *key_ptr, const int keybuf_size, const std::vector<int> &vec_field_no, char *field_buf_ptr, const int buf_size);

A.5.16.2 接口说明：该接口用于根据传入多条记录的关键字和需要查询的域号，查询出相应记录的域值。如果查询成功，则结果返回大于等于 0；如果查询失败结果返回小于 0。

A.5.16.3 参数列表见表 A.104：

表 A.104 TableGetByKey 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
key_ptr	char*	In	支持查询单个关键字或多个关键字的实时库表，每行记录的关键字通过 memcpy 按序赋值到 key_ptr 内存段中，每次 memcpy 拷贝关键字数据类型或结构体的 sizeof 大小，详见接口示例	是
keybuf_size	const int	In	表明 key_ptr 关键字总大小，即为 memcpy 的总大小	是
vec_field_no	const std::vector<int>&	In	查询的域号	是
field_buf_ptr	char*	Out	存放取出数据的指针，需要调用者预先分配好内存	是
buf_size	const int	In	参数 field_buf_ptr 分配内存的大小	是
返回值	Int	Out	>=0 返回记录条数 <0 接口调用失败	—

A.5.16.4 接口示例：

（CJKTableOp 本地接口访问）：

```
int main()
{
    //示例说明 1：单关键字表数据查询
    // 1）单个关键字 405 厂站表查询，关键字数据类型为 long 型
    // 2）指定两条记录关键字 id，查询厂站表 2、3 号域（code、name）数据
    struct field_stru{
        char code[32];
        char name[64];
    };

    int app_no = 100000;
    int context_no = 1;
    int table_no = 405;
```

```
CJKTableOp table_operation_obj;
int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
if(ret_code != 0)
{
    return ret_code;
}

std::vector<int> vec_field_no(2);  //查询域个数
vec_field_no[0] = 2;  //查询域号
vec_field_no[1] = 3; //查询域号
int buf_size = sizeof(field_stru) * 2;
char* field_buf_ptr = (char*)malloc(buf_size*sizeof(char));

long keyid1 = 113997365567815682;
long keyid2 = 113997365567815683;
int keybuf_size = 2 * sizeof(long); //关键字数据类型大小
char* key_ptr = (char*)malloc(keybuf_size);  // 创建存放 2 条记录关键字的内存段
memcpy(key_ptr, (char *)&keyid1 ,  sizeof(long));
memcpy(key_ptr + sizeof(long), (char *)&keyid2 ,  sizeof(long)); //关键字按序赋值

ret_code = table_operation_obj.TableGetByKey (key_ptr, keybuf_size, vec_field_no, field_buf_ptr,
    buf_size);
if(ret_code < 0)
{
    return ret_code;
}
free(key_ptr);
free(field_buf_ptr);

//示例说明 2：多关键字表数据查询
// 1）多个关键字 1990 号测试表查询，该表为 long 型 id 和 time_t 时间类型 datetime 的复合关键字，建议关键字域长度为 8 的倍数
// 2）使用结构体存放关键字信息
// 3）查询该表所有域的域值
struct key_stru{
    long id;
```

```
        time_t datetime;
    };

    table_no = 1990;
    ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code != 0)
    {
        return ret_code;
    }

    //获取记录长度
    short field_num = 0;
    int record_num = 0;
    int record_size = 0; //获取整行记录数据长度
    ret_code = table_operation_obj.GetTablePara(field_num, record_num, record_size);
    if(ret_code != 0)
    {
        return ret_code;
    }

    key_stru key;
    key.id = 113997365567815682;
    key.datetime = 1676291131;
    std::vector<int> vec_field_all(1);
    vec_field_all[0] = -1; //vec_field_all size = 1 且值为-1，表示取整条记录所有域
    char* rec_ptr = (char*)malloc(record_size);
    ret_code = table_operation_obj.TableGetByKey ((char *)&key, sizeof(key_stru), vec_field_all, rec_ptr,
record_size);
    if(ret_code < 0)
    {
        return ret_code;
    }
    free(rec_ptr);

    return 0;
}
```

（CJKTableNet 网络接口访问）：

```
int main()
{
    //示例说明 1：单关键字表数据查询
    // 1）单个关键字 405 厂站表查询，关键字数据类型为 long 型
    // 2）指定两条记录关键字 id，查询厂站表 2、3 号域（code、name）数据
    struct field_stru{
        char code[32];
        char name[64];
    };

    int app_no = 100000;
    int context_no = 1;
    int table_no = 405;
    CJKTableNet table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code != 0)
    {
        return ret_code;
    }

    std::vector<int> vec_field_no(2);
    vec_field_no[0] = 2;
    vec_field_no[1] = 3;
    int buf_size = sizeof(field_stru) * 2;
    char* field_buf_ptr = (char*)malloc(buf_size*sizeof(char));

    long keyid1 = 113997365567815682;
    long keyid2 = 113997365567815683;
    int keybuf_size = 2 * sizeof(long);
    char* key_ptr = (char*)malloc(keybuf_size);
    memcpy(key_ptr, (char *)&keyid1 ,  sizeof(long));
    memcpy(key_ptr + sizeof(long), (char *)&keyid2 ,  sizeof(long));

    ret_code = table_operation_obj.TableGetByKey (key_ptr, keybuf_size, vec_field_no, field_buf_ptr,
buf_size);
     if(ret_code < 0)
```

```
{
    return ret_code;
}
free(key_ptr);
free(field_buf_ptr);

//示例说明 2：多关键字表数据查询
// 1）多个关键字 1990 号测试表查询，该表为 long 型 id 和 time_t 时间类型 datetime 的复合关键字，建议关键字域长度为 8 的倍数
// 2）使用结构体存放关键字信息
// 3）查询该表所有域的域值
struct key_stru{
    long id;
    time_t datetime;
};

table_no = 1990;
ret_code = table_operation_obj.Open(app_no, table_no, context_no);
if(ret_code != 0)
{
    return ret_code;
}

//获取记录长度
short field_num = 0;
int record_num = 0;
int record_size = 0;
ret_code = table_operation_obj.GetTablePara(field_num, record_num, record_size);
if(ret_code != 0)
{
    return ret_code;
}

key_stru key;
key.id = 113997365567815682;
key.datetime = 1676291131;
std::vector<int> vec_field_all(1);
```

```
    vec_field_all[0] = -1; //vec_field_all size = 1 且值为-1，表示取整条记录
    char* rec_ptr = (char*)malloc(record_size);
    ret_code = table_operation_obj.TableGetByKey ((char *)&key, sizeof(key_stru), vec_field_all, rec_ptr,
record_size);
  if(ret_code < 0)
    {
        return ret_code;
    }
    free(rec_ptr);

    return 0;
}
```

A.5.17 根据多条记录的关键字修改多个域的域值

A.5.17.1 接口原型： int TableModifyByKey(const char *key_ptr, const int keybuf_size, const std::vector<int> &vec_field_no, const char *field_buf_ptr, const int buf_size);

A.5.17.2 接口说明：该接口用于根据传入多条记录的关键字和需要修改的域号，修改相应记录的域值。如果修改成功，则结果返回修改成功个数；如果查询失败结果返回小于 0。

A.5.17.3 参数列表见表 A.105：

表 A.105 TableModifyByKey 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
key_ptr	char*	In	支持修改单个关键字或多个关键字的实时库表，每行记录的关键字通过 memcpy 按序赋值到 key_ptr 内存段中，每次 memcpy 拷贝关键字数据类型或结构体的 sizeof 大小，详见接口示例	是
keybuf_size	const int	In	表明 key_ptr 关键字总大小，即为 memcpy 的总大小	是
vec_field_no	const std::vector<int>&	In	修改的域号	是
field_buf_ptr	const char*	In	存放修改数据的指针	是
buf_size	const int	In	参数 field_buf_ptr 分配内存的大小	是
返回值	Int	Out	>=0 修改成功个数 <0 接口调用失败	—

A.5.17.3 接口示例：

（CJKTableOp 本地接口访问）：

```
int main()
{
    //示例说明 1：单关键字表数据修改
    // 1）单个关键字 405 厂站表修改，关键字数据类型为 long 型
```

```
// 2）指定两条记录关键字 id，修改厂站表 2、3 号域（code、name）数据
struct field_stru{
	char code[32];
	char name[64];
};

int app_no = 100000;
int context_no = 1;
int table_no = 405;
CJKTableOp table_operation_obj;
int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
if(ret_code != 0)
{
	return ret_code;
}

std::vector<int> vec_field_no(2); //修改域个数
vec_field_no[0] = 2; //修改域号
vec_field_no[1] = 3; //修改域号

//设置修改的域值
int buf_size = sizeof(field_stru) * 2;
char* field_buf_ptr = (char*)malloc(buf_size*sizeof(char));
field_stru field1;
sprintf(field1.code, "%s", "code1_modify");
sprintf(field1.name, "%s", "name1_modify");
field_stru field2;
sprintf(field2.code, "%s", "code2_modify");
sprintf(field2.name, "%s", "name2_modify");
memcpy(field_buf_ptr, (char*)&field1, sizeof(field_stru));
memcpy(field_buf_ptr + sizeof(field_stru), (char*)&field2, sizeof(field_stru));

//设置关键字的值
long keyid1 = 113997365567815682;
long keyid2 = 113997365567815683;
int keybuf_size = 2 * sizeof(long); //关键字数据类型大小
char* key_ptr = (char*)malloc(keybuf_size); // 创建存放 2 条记录关键字的内存段
```

```
memcpy(key_ptr, keyid1 ， sizeof(long));
memcpy(key_ptr + sizeof(long), keyid2 ， sizeof(long));


ret_code = table_operation_obj.TableModifyByKey ((char *)&key_ptr, keybuf_size, vec_field_no,
field_buf_ptr, buf_size);
if(ret_code < 0)
{
    return ret_code;
}
free(key_ptr);
free(field_buf_ptr);


//示例说明 2：多关键字表数据修改
// 1）多个关键字 1990 号测试表修改，该表为 long 型 id 和 time_t 时间类型 datetime 的复合关键
字，建议关键字域长度为 8 的倍数
// 2）使用结构体存放关键字信息
// 3）修改该表的 st_id 字段
struct key_stru{
    long id;
    time_t datetime;
    };


table_no = 1990;
ret_code = table_operation_obj.Open(app_no, table_no, context_no);
if(ret_code != 0)
{
    return ret_code;
}


key_stru key;
key.id = 113997365567815682;
key.datetime = 1676291131;
std::vector<int> vec_field_no2(1);
vec_field_no2[0] = 5;
long st_id = 113997365567815683;
ret_code = table_operation_obj.TableModifyByKey ((char *)&key, sizeof(key_stru), vec_field_no2,
(char*)&st_id, sizeof(long));
```

```
    if(ret_code < 0)
    {
        return ret_code;
    }

    return 0;
}
```

（CJKTableNet 网络接口访问）：

```
int main()
{
    //示例说明 1：单关键字表数据修改
    // 1）单个关键字 405 厂站表修改，关键字数据类型为 long 型
    // 2）指定两条记录关键字 id，修改厂站表 2、3 号域（code、name）数据
    struct field_stru{
        char code[32];
        char name[64];
    };

    int app_no = 100000;
    int context_no = 1;
    int table_no = 405;
    CJKTableOp table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code != 0)
    {
        return ret_code;
    }

    std::vector<int> vec_field_no(2);
    vec_field_no[0] = 2;
    vec_field_no[1] = 3;

    //设置修改的域值
    int buf_size = sizeof(field_stru) * 2;
    char* field_buf_ptr = (char*)malloc(buf_size*sizeof(char));
    field_stru field1;
```

```
sprintf(field1.code, "%s", "code1_modify");
sprintf(field1.name, "%s", "name1_modify");
field_stru field2;
sprintf(field2.code, "%s", "code2_modify");
sprintf(field2.name, "%s", "name2_modify");
memcpy(field_buf_ptr, (char*)&field1, sizeof(field_stru));
memcpy(field_buf_ptr + sizeof(field_stru), (char*)&field2, sizeof(field_stru));

//设置关键字的值
long keyid1 = 113997365567815682;
long keyid2 = 113997365567815683;
int keybuf_size = 2 * sizeof(long);
char* key_ptr = (char*)malloc(keybuf_size);
memcpy(key_ptr, keyid1 ， sizeof(long));
memcpy(key_ptr + sizeof(long), keyid2 ， sizeof(long));

ret_code = table_operation_obj.TableModifyByKey ((char *)&key_ptr, keybuf_size, vec_field_no,
field_buf_ptr, buf_size);
if(ret_code < 0)
{
    return ret_code;
}
free(key_ptr);
free(field_buf_ptr);

//示例说明 2：多关键字表数据修改
// 1）多个关键字 1990 号测试表修改，该表为 long 型 id 和 time_t 时间类型 datetime 的复合关键
字，建议关键字域长度为 8 的倍数
// 2）使用结构体存放关键字信息
// 3）修改该表的 st_id 字段
struct key_stru{
    long id;
    time_t datetime;
};

table_no = 1990;
ret_code = table_operation_obj.Open(app_no, table_no, context_no);
```

```
    if(ret_code != 0)
    {
        return ret_code;
    }

    key_stru key;
    key.id = 1139973655678156 82;
    key.datetime = 1676291131;
    std::vector<int> vec_field_no2(1);
    vec_field_no2[0] = 5;
    long st_id = 1139973655678156 83;
    ret_code = table_operation_obj.TableModifyByKey ((char *)&key, sizeof(key_stru), vec_field_no2,
(char*)&st_id, sizeof(long));
    if(ret_code < 0)
    {
        return ret_code;
    }

    return 0;
}
```

A.5.18 根据关键字查询级联关系名称

A.5.18.1 接口原型：int GetNameStringByID(const long reference_id, std::string &ref_string);

A.5.18.2 接口说明：该接口支持根据传入的设备、量测等关键字 ID（该 ID 域的数据类型是 long 型，关键字生成类型是按表号自动生成），转译出对应的级联关系名称。接口调用成功则返回大于等于 0；失败则返回小于 0。

A.5.18.3 参数列表见表 A.106：

表 A.106 GetNameStringByID 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
reference_id	const long	In	关键字	是
ref_string	std::string&	Out	关键字的级联关系名称	是
返回值	Int	Out	>=0 成功 <0 失败	—

A.5.18.4 接口示例：

（CJKApiOp 本地接口访问）：

```
    int main()
    {
        int app_no = 1600000;
        int context_no = 1;  //实时态
        int table_no = 1;
        CJKApiOp api_operation_obj;
```

```
        int ret_code =api_operation_obj.SetAppNo(app_no,context_no);
        if(ret_code <0)
        {
            return ret_code;
        }

        long key_id=113997365567815682;  //设备 id
        std::string name_string;
        ret_code = api_operation_obj.GetNameStringByID (key_id, name_string);
        printf("ret_code=%d,name_string=%s\n", ret_code, name_string.c_str());


         if(ret_code < 0)
        {
            return ret_code;
        }
        else
        {
          //打印域名
        }
        return 0;
    }
```

（CJKApiNet 网络接口访问）：

```
    int main()
    {
        int app_no = 1600000;
        int context_no = 1;  //实时态
        int table_no = 1;
        CJKTableNet table_operation_obj;
        CJKApiNet api_operation_obj;
        int ret_code =api_operation_obj.SetAppNo(app_no,context_no);
        if(ret_code <0)
        {
            return ret_code;
        }

        long key_id=113997365567815682;
        std::string name_string;
        ret_code = api_operation_obj.GetNameStringByID (key_id, name_string);
        if(ret_code < 0)
        {
            return ret_code;
        }
        else
        {
          //打印域名
        }
        return 0;
    }
```

A.5.19 获取多个域的类型、长度和偏移量

A.5.19.1 接口原型：int GetFieldParaByNo(std::vector<MEMBER_DEFINITION_JK> &vec_offset, const std::vector<int> &vec_field_no);

A.5.19.2 接口说明：该接口主要用于获取多个域的类型、长度和偏移量。其中 vec_offset 的 size 比 vec_field_no 的 size 大 1。且出参 vec_offset 最后一个元素的_data_offset 是对齐后的域长度之和。接口调用成功则返回 0；失败则返回<0。

A.5.19.3 参数列表见表 A.107：

表 A.107 GetFieldParaByNo 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
vec_offset	std::vector<MEMBER_DEFINITION_JK> &	Out	返回多个域的类型、长度和偏移量	是
vec_field_no	const std::vector<int> &	In	查询的多个域号	是
返回值	Int	Out	=0 成功 <0 失败	—

A.5.19.3 参数说明：

```
struct MEMBER_DEFINITION_JK
{
    int data_index;      //域号
    int data_type;       //域类型
    int data_size;       //域长度
    int data_offset;     //域偏移
};
```

A.5.19.4 接口示例：

（CJKTableOp 本地接口访问）：

```
int main()
{
    int app_no = 1600000;
    int context_no = 1;  //实时态
    int table_no = 1;
    CJKTableOp table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
        return ret_code;
    }

  std::vector<int> vec_field_no;
    vec_field_no.push_back(2);
    vec_field_no.push_back(3);
  std::vector<MEMBER_DEFINITION_JK> vec_offset;
    ret_code = table_operation_obj.GetFieldParaByNo (vec_offset, vec_field_no);
  if(ret_code < 0)
    {
    return ret_code;

    }
    else
    {
        for (unsigned int i = 0; i < vec_field_no.size() - 1; ++i)
```

```
        {
            //输出字段 data_type、data_size、data_offset 等信息
        }
    }
    return 0;
}
```

（CJKTableNet 网络接口访问）：

```
int main()
{
    int app_no = 1600000;
    int context_no = 1;  //实时态
    int table_no = 1;
    CJKTableNet table_operation_obj;
    int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
    if(ret_code <0)
    {
        return ret_code;
    }

  std::vector<int> vec_field_no;
  vec_field_no.push_back(2);
    vec_field_no.push_back(3);
  std::vector<MEMBER_DEFINITION_JK> vec_offset;
    ret_code = table_operation_obj.GetFieldParaByNo (vec_offset, vec_field_no);
  if(ret_code < 0)
    {
        return ret_code;
    }
    else
    {
      for (unsigned int i = 0; i < vec_field_no.size() - 1; ++i)
      {
          //输出字段 data_type、data_size、data_offset 等信息
      }
    }
    return 0;
}
```

A.5.20 获取表信息

A.5.20.1　接口原型：int GetTablePara(short &field_num, int &record_num, int &record_size)

A.5.20.2　接口说明：该接口用于获取实时库表中的记录个数、域个数和一行记录所有域长度和。

A.5.20.3　参数列表见表 A.108：

表 A.108 GetTablePara 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
field_num	Short	Out	表域个数	是
record_num	Int	Out	表记录数	是
record_size	Int	Out	一行记录所有域长度	是
返回值	Int	Out	>=0　成功 <0　失败	—

A.5.20.4 接口示例：

（CJKTableOp 本地接口访问）：

```
int main()
{
	int app_no = 100000;
	int context_no = 1;  //实时态
	int table_no = 220;
	CJKTableOp table_operation_obj;
	int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
	if(ret_code <0)
	{
		return ret_code;
	}
	short field_num(0) ;
	int record_num(0) ;
	int record_size(0) ;
	ret_code = table_operation_obj.GetTablePara(field_num, record_num, record_size);
	if(ret_code <0)
	{
		return ret_code;
	}
	else
	{
	  //打印域信息
	}
	return 0;
}
```

（CJKTableNet 网络接口访问）：

```
int main()
{
	int app_no = 100000;
	int context_no = 1;  //实时态
	int table_no = 220;
	CJKTableNet table_operation_obj;
	int ret_code = table_operation_obj.Open(app_no, table_no, context_no);
	if(ret_code <0)
	{
		return ret_code;
	}
	short field_num(0) ;
	int record_num(0) ;
	int record_size(0) ;
	ret_code = table_operation_obj.GetTablePara(field_num, record_num, record_size);


	if(ret_code <0)
	{
		return ret_code;
	}
	else
	{
	  //打印域信息
	}
```

```
    return 0;
}
```

A.6 历史数据服务接口

A.6.1 基本要求

基础平台提供统一的历史数据访问接口，通过调用这些接口访问平台内集成的历史数据服务，并返回执行结果。接口头文件及动态库名称见基础说明 2.8。

A.6.2 获得曲线数据

A.6.2.1 接口原型：int GetCurveData(JKCurvePara &param, time_t timeout, JKCurveData** datap,int&items, JKErrorInfo&error)

A.6.2.2 接口说明:该接口客户端调用，根据传入的参数查询曲线数据，如果任务完成返回成功，否则返回失败。参数列表见表 A.109：

表 A.109 GetCurveData 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
Param	JKCurvePara	In/Out	曲线参数信息结构，包含应用号，测点的 key_id，曲线查询的起止时间，查询的步长，查询的策略等信息	是
timeout	time_t	In	因是同步，这是接口设置的返回的超时时间	是
datap	JKCurveData **	In	返回数据的指针	是
items	int	Out	曲线返回数据的个数	是
error	JKErrorInfo	Out	出错代码的返回结果	是
返回值	int	Out	>=0：成功；-1：失败	是

A.6.2.3 参数说明：

```
struct JKCurvePara
{
  char    conf_id[128];      //采样表定义的配置号码。基于原始数据查询，conf_id = “HIS_O_DATA”；基于日统计数据查询，conf_id = “HIS_DAY_MAX/HIS_YAER_MAX”（HIS_DAY_MIN、HIS_DAY_AVG）等
  time_t   starttime;  //要求取得采样的起始时间
  time_t   stoptime;   //要求取得采样的终止时间
  char   dev_key_id_values[128];   //指定要取得采样设备 ID，该 ID 的字符串长度不能超过 127 字节
  int   needpace;   //采样的时间步长，如果指定的是 0，那么按配置文件中配置的步长为默认值
  int    data_format;   //查询策略 1：最大值；2：最小值；3：平均值；4：第一个值；5：最后一个值
  void*   temp_memory1;  //保留字段
  void*   temp_memory2;  //保留字段
  char   resververd[32]; //保留空间
};

struct JKCurveData
{
    int    is_null;   //是否是空值
```

```
        int     state;    //采样设备在该时间的状态位，如果无状态位，该成员值是无效的
        union
        {
          float f_value;   //采样值
          unsigned int i_value; //采样值,这种类型有采样值只支持 FIRST_VALUE, LAST_VALUE
取数
        };
        union
        {
          time_t x_time;//采样的时间
          float   x_value;
        };
    };

    struct JKErrorInfo
    {
      int error_no;
      char error_info[JK_ERROR_INFO_LEN + 1];
      int file_line;
      char file_name[300];
    };
```

1） 基于原始数据的曲线查询

JKCurvePara.conf_id = “HIS_O_DATA”；

needpace：曲线查询步长（单位秒），查询时步长 >= 采样周期。

举例：查询 1 分钟周期的采样数据，按 5 分钟步长取数。返回数据集合按查询开始时间和结束时间范围内每 5 分钟返回 1 个数据值。当查询策略为最大值时，每 5 分钟内返回 5 个采样值中的最大值。查询条件，2023-01-01 00:00:00 至 2023-01-01 00:09:59，JKCurvePara.conf_id = “HIS_O_DATA”， needpace = 5，策略最大值。返回数据：2 个值。

2） 基于日统计数据查询

JKCurvePara.conf_id 取值包括：HIS_DAY_MAX-日最大值、 HIS_DAY_MIN-日最小值、HIS_DAY_AVG 日平均值、HIS_MONTH_MAX-月最大值、 HIS_MONTH_MIN-月最小值、HIS_MONTH_AVG 月平均值、HIS_YER_MAX-年最大值、 HIS_YEAR_MIN-年最小值、HIS_YEAR_AVG 年平均值。该方式查询时不即时统计，而是返回查询时间范围内日统计结果的数据再二次计算。

举例：按开始时间和结束时间，以日统计数据为基础数据返回月、年统计数据。查询条件，2022-01-01 00:00:00 至 2022-12-31 23:59:59，JKCurvePara.conf_id = “HIS_DAY_MAX”。返回数据：返回 2022-01-01 00:00:00 至 2022-12-31 23:59:59 范围内每日的最大值，共计 365 个数据点；JKCurvePara.conf_id = “HIS_MONTH_MAX”。返回数据：返回 2022-01-01 00:00:00 至 2022-12-31 23:59:59 范围内每月的最大值，共计 12 个数据点；JKCurvePara.conf_id = “HIS_YEAR_MAX”。返回数据：返回 2022-01-01 00:00:00 至 2022-12-31 23:59:59 范围内年的最大值，共计 1 个数据点。

表 A.110 JKCurvePara.conf_id 参数列表

查询场景	值	说明

基于原始数据查询	HIS_O_DATA	返回原始采样数据
基于日统计数据查询	HIS_DAY_MAX	返回每日的最大值
	HIS_DAY_MIN	返回每日的最小值
	HIS_DAY_AVG	返回每日的平均值
	HIS_MONTH_MAX	返回每月的最大值
	HIS_ MONTH _MIN	返回每月的最小值
	HIS_ MONTH _AVG	返回每月的平均值
	HIS_YEAR_MAX	返回每年的最大值
	HIS_ YEAR _MIN	返回每年的最小值
	HIS_ YEAR _AVG	返回每年的平均值

A.6.2.4 接口示例：

```
using namespace JK_API;

CJKHisServiceClientInterface *his_client = new CJKHisServiceClientInterface();
int times = 0;
JKCurvePara param; // 查询曲线结构
memset (&param, 0, sizeof (struct CurvePara));
strcpy (param.conf_id, "1"); // 针对曲线查询场景，入参 1
param.data_format = 1;//查询策略：1：最大值；2：最小值；3：平均值；4：第一个值；5：最后一个值
param.needpace = 300;  //无配置文件，由调用方根据实际需求指定，采样的时间步长 300 秒，是查询测点的采样周期的倍数。
param.starttime = start;  // 查询的开始时间
param.stoptime = finish; //  查询的结束时间
strcpy (param.dev_key_id_values, "38004753993354980 51");  //查询测点的 keyid
JKCurveData *datapp = NULL;
int items = 0;
JKErrorInfo error;  // 返回错误信息
int ret = 0;
ret = his_client->GetCurveData(param, 20, &datapp, items, error)
if (ret)  //成功返回 1.
{
    cout << "GetCurveData 操作成功, 返回" << items << "个采样点" << endl;
    for (int i = 0; i < items; i++)
    {
        cout<<"GetCurveData, 采样点"<< i << ", 是否空值: " << datapp[i].is_null << ", 状态: " << datapp[i].state << ", 采样值: " << datapp[i].f_value << endl;
    }
}
else
{
    cout<<"GetCurveData 操作失败, 返回值 ret ="<<ret <<", 错误号: "<<error.error_no<<", 错误描述:" << error.error_info << endl;
}

free (datapp);
delete his_client;
his_client = NULL;
```

A.7 商用库服务接口

A.7.1 基本要求

应用通过本节的接口调用可以实现通过指定 sql 同步/异步对商用数据库执行增、删、改操作的功能。

建立直接 SQL 服务客户端时必须构造一个 CJKSqlSpClient 类的对象，由该类的接口负责与直接 SQL 服务端通信。该类主要提供一系列的接口函数完成相关数据查询和修改的功能。接口头文件及动态库名称见基础说明 2.8。

A.7.2 数据查询接口

A.7.2.1 接口原型：： short SelectSql (const char* sql_str, unsigned int max_result_num, SEQOutDataTypeStru& out_data_type, TSelectResultStru& out_select_result,SEQDBErrorStru& out_db_error)。

A.7.2.2 接口说明:该接口客户端调用，通过传入查询类型的 sql 语句服务执行任务，任务执行完成返回成功，否则返回失败。

A.7.2.3 参数列表见表 A.111：

表 A.111 SelectSql 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
const char * sql_str	char*	In	需要查询的 SQL 语句	是
unsigned int max_result_num	unsigned int	In	最大的返回记录数	是
SEQOutDataTypeStru& out_data_type	SEQOutDataTypeStru	In	对传出结果每个域的数据类型定义	是
TSelectResultStru& out_select_result	TSelectResultStru	Out	查询结果的返回结构	是
SEQDBErrorStru& out_db_error	SEQDBErrorStru	Out	出错代码的返回结果	是
返回值	short	Out	0：查询成功 -1：查询失败	是

A.7.2.4 参数说明：

```
struct TOutDataTypeStru
{
   short   out_field_position; // 在查询语句中 SELECT 子句的位置，从 1 开始
   short   out_field_datatype; // 返回参数的数据类型
};
typedef vector<TOutDataTypeStru> SEQOutDataTypeStru;
//数据类型包括
const short C_DATATYPE_STRING = 1;
const short C_DATATYPE_UCHAR = 2;
const short C_DATATYPE_SHORT = 3;
const short C_DATATYPE_INT = 4;
const short C_DATATYPE_DATETIME = 5;
const short C_DATATYPE_FLOAT = 6;
const short C_DATATYPE_DOUBLE = 7;
const short C_DATATYPE_LONG = 15;


// SELECT 请求返回的报文结构
struct TSelectResultStru
{
   short  field_num;  // 查询的域的个数
```

```
    SEQResultFieldInfo  field_info; // 每个域的基本信息
    unsigned int  data_num; // 查询结果的个数
    SEQResultDataValue data_value_seq; // 每个查询结果的值，按照先行后列的顺序排列
};

// 出错信息结构
struct TDBErrorStru
{
    unsigned int     error_no; // 错误号
    MLang::STRING        error_msg; // 错误信息
};

// 出错信息结构序列
typedef vector<TDBErrorStru> SEQDBErrorStru;
```

接口返回值:

返回值	说明	备注
>=0	成功	—
<0	失败	—

A.7.2.5 接口示例：

查询 hawk_test2 表中的数据，表 hawk_test2 结构如下：

```
create table hawk_test2{
long col1;
int int_test;
}
```

接口调用方法如下：

```
/*
CJKSqlResultAlignClient 类包含的方法：GetAlignResultClient
函数原型：
int GetAlignResultClient(SEQResultFieldInfo &field_ptr, SEQResultDataValue &data_ptr, VIndicator &null_vec, void *align_result_ptr, int struct_length);

函数功能：查询结果按照数据结构 TQueryResult 进行数据对齐
输入参数 ：  field_ptr          域的信息
        data_ptr          结果信息，按照先行后列的顺序排列
        struct_length       目标结构的大小，用 sizeof 来调用
输出参数 ：  null_vec          空值标志序列，TRUE 表示 NULL，FALSE 表示非空
                    其长度为 data_ptr->length()，按照先行后列的顺序排列
        align_result_ptr     目标结构的指针，需要在调用时分配正确的空间
返回值 ：  < 0            失败
        > 0            成功
*/
using namespace JK_API;
// 通过构造函数的入参端口，确定连接商用库服务。
CJKSqlSpClient *sql_sp_client = new CJKSqlSpClient ();
char  sql_str[3000];
int ret=0;
CJKSqlResultAlignClient sql_align; // 数据对齐
SEQOutDataTypeStru out_data_type; // 对传出结果的域的数据类型定义
TSelectResultStru out_select_result; // 查询结果返回结果
```

```
SEQDBErrorStru out_db_error; // 出错的时候返回错误信息
strcpy(sql_str, "select col1, int_test from hawk_test2"); // 执行查询 sql 语句
retcode = sql_sp_client->SelectSql(sql_str, QUERY_ALL_RESULT, out_data_type, data_result,
out_db_error);
if (retcode == 0)
{
    cout << "SelectSql  执行成功" << endl;
cout << "SelectSql------>column size = " << out_data_type.size() << endl;

// 打印返回结果的域信息
    for (unsigned int i = 0; i < out_data_type.length(); i++)
    {
        cout << "column[ " << i << " ] type = " << out_data_type[i].out_field_datatype << endl;
    }

    if (out_select_result.data_num <= 0) // 判断返回的记录数
    {
        delete sql_sp_client;
        sql_sp_client = NULL;
        return;
    }
    else
    {
// 返回数据结构体
        struct TQueryResult
        {
            long col1;
            int int_test;
        };

        TQueryResult *query_result_ptr; // 查询结果的数据指针

        query_result_ptr = new TQueryResult[out_select_result.data_num];
        ret = sql_align.GetAlignResultClient (out_select_result.field_info,
        out_select_result.data_value_seq, null_vec,
        query_result_ptr,sizeof (TQueryResult)); // 查询结果按照数据结构 TQueryResult 进行数据对齐

        if (ret != 1) // 数据对齐失败
        {
            cout << "SelectSql------>Align error." << endl;
            delete sql_sp_client;
            sql_sp_client = NULL;
            return;
        }
        else
        {
            // 打印查询的记录
            for (unsigned int i = 0; i < out_select_result.data_num; i++)
            {
                cout <<"SelectSql------>result[" << i << "], col1 = " << query_result_ptr[i].col1 << endl;
            }
            delete[] query_result_ptr;
            }
        }
    }
```

```
}else{
cout << "SelectSql 执行失败" << endl;
      for (unsigned int i = 0; i < out_db_error.length(); i++)
      {
            char str[256];
            sprintf(str,"program with error: err_no = %d, err_msg = %s\n",(int) out_db_error[i].error_no, (char
*) out_db_error[i].error_msg);
            cout<< str << endl;
      }
}

delete sql_sp_client;
sql_sp_client = NULL;
```

A.7.3 数据修改接口

A.7.3.1 接口原型：short ExecuteSql (const char* sql_str, SEQDBErrorStru& out_db_error)。

A.7.3.2 接口说明：该接口客户端调用，通过传入修改类型的 sql 语句服务执行任务，任务执行完成返回成功，否则返回失败。

A.7.3.3 参数列表见表 A.112：

表 A.112 ExecuteSql 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
const char* sql_str	char*	In	修改的 SQL 语句	是
SEQDBErrorStru& out_db_error	SEQDBErrorStru	Out	出错代码的返回结果	是
返回值	short	Out	0：修改成功 -1：失败	是

A.7.3.4 参数说明：

```
// 出错信息结构
struct TDBErrorStru
{
   unsigned int     error_no; // 错误号
   MLang::STRING         error_msg; // 错误信息
};

// 出错信息结构序列
typedef vector<TDBErrorStru> SEQDBErrorStru;
```

A.7.3.5 接口返回值：

返回值	说明	备注
=0	成功	—
<0	失败	—

A.7.3.6 接口示例：

```
using namespace JK_API;
CJKSqlSpClient *sql_sp_client = new CJKSqlSpClient ();
SEQDBErrorStru_var   seq_db_err; // 如果执行失败返回错误信息
char  sql_str[3000];
strcpy(sql_str, "insert into substation(id,code,name)
values(113997365567815799,'hawk_test','hawk_test')"); // 查询的 sql 语句
int retcode = sql_sp_client->ExecuteSql(sql_str, seq_db_err);
```

```
if (retcode == 0)
{
  cout << "ExecuteSql 执行成功" << endl;
}else{
      cout << "ExecuteSql 执行失败" << endl;
      for (unsigned int i = 0; i < out_db_error.length(); i++)
      {
            char str[256];
            sprintf(str,"program with error: err_no = %d, err_msg = %s\n",(int)
out_db_error[i].error_no, (char *) out_db_error[i].error_msg);
            cout<< str << endl;
      }
}

delete sql_sp_client;
sql_sp_client = NULL;
```

A.7.4 批量数据修改接口

A.7.4.1 接口原型：short ExecuteMultiSql(int sql_num,const StrSeq& seq_sql_str, SEQExecMultiResultStru& seq_exec_multi_result);

A.7.4.2 接口说明：该接口客户端调用，通过传入多条修改类型的 sql 语句服务执行任务，任务执行完成返回成功，否则返回失败。

A.7.4.3 参数列表见表 A.113：

表 A.113 ExecuteMultiSql 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
int sql_num	int	In	需执行的修改 SQL 语句总数	是
const StrSeq& seq_sql_str	StrSeq	In	记录多条修改 SQL 语句的结构	是
SEQExecMultiResultStru & seq_exec_multi_result	SEQExecMultiResultStru	Out	记录每条 SQL 语句修改结果的结构	是
返回值	int	Out	0：查询成功 -1：查询失败	是

A.7.4.4 参数说明：

```
// 记录多条修改 SQL 语句的结构
typedef MLang::VECTOR<MLang::STRING> StrSeq;

// 出错信息结构
struct TDBErrorStru
{
  unsigned int     error_no; // 错误号
  MLang::STRING        error_msg; // 错误信息
};

struct TExecMultiResultStru
{
      bool is_success;
      TDBErrorStru db_error;
};

// 记录每条 SQL 语句修改结果的结构
```

typedef VECTOR<TExecMultiResultStru> SEQExecMultiResultStru;

A.7.4.5 接口示例：

```
using namespace JK_API;
CJKSqlSpClient *sql_sp_client = new CJKSqlSpClient ();
SEQExecMultiResultStru  seq_db_str;
StrSeq  str_seq;
char  sql_str[3000];

strcpy(sql_str, "insert into substation(id,code,name)
values(113997365567815799,'hawk_test','hawk_test')");  // 第 1 条修改的 sql 语句
str_seq.push_back(sql_str);

strcpy(sql_str, "insert into substation(id,code,name)
values(113997365567815800,'hawk_test1','hawk_test1')");  // 第 2 条修改的 sql 语句
str_seq.push_back(sql_str);

strcpy(sql_str, "insert into substation(id,code,name)
values(113997365567815801,'hawk_test2','hawk_test2')");  // 第 3 条修改的 sql 语句
str_seq.push_back(sql_str);

int retcode = sql_sp_client->ExecuteMultiSql(3,str_seq,seq_db_str);
if (retcode == 0)
{
  cout << "ExecuteMultiSql 执行成功" << endl;
}else{
      cout << "ExecuteMultiSql 执行失败" << endl;
}

delete sql_sp_client;
sql_sp_client = NULL;
```

A.7.5 绑定变量数据修改接口

A.7.5.1 接口原型：short ExecuteBind(const char * sql_str, SEQBindParaStru &seq_bind_para, char * data_buffer, char * null_flag_buffer, unsigned int row_num, SEQDBErrorStru& out_db_error);

A.7.5.2 接口说明：该接口客户端调用，通过传入绑定变量形式的 sql 语句执行任务，任务执行完成返回成功，否则返回失败。

A.7.5.3 参数列表见表 A.114：

表 A.114 ExecuteBind 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
sql_str	const char*	In	提交的 sql 语句	是
seq_bind_para	SEQBindParaStru&	In	绑定变量参数结构	是
data_buffer	char *	In	写入数据流	是
null_flag_buffer	char *	In	空标志位流	是
row_num	int	In	写入数据行数	是
out_db_error	SEQDBErrorStru&	Out	执行报错信息	是
返回值	int	Out	1：提交成功 -1：提交失败	是

A.7.5.4 参数说明：

```
// 绑定变量参数结构
struct TBindParaStru
{
    unsigned short dci_type;
    unsigned int    data_size;
    void __write(MLang::OutputStream&__os)const;
    void __read(MLang::InputStream&__is);
};
typedef MLang::VECTOR<TBindParaStru> SEQBindParaStru;

// 执行报错信息结构
struct TDBErrorStru
{
    unsigned int    error_no;
    MLang::STRING   error_msg;
    TDBErrorStru();
    TDBErrorStru(const TDBErrorStru&);
    TDBErrorStru&operator=(const TDBErrorStru&);
    void __write(MLang::OutputStream&__os)const;
    void __read(MLang::InputStream&__is);
};
typedef MLang::VECTOR<TDBErrorStru> SEQDBErrorStru;
```

A.7.5.5 接口示例：

```
using namespace JK_API;
CJKSqlSpClient *sql_sp_client = new CJKSqlSpClient ();
string bind_str = "insert into rdb_test_table1 (id, test_name, test_age, test_date, test_float) values (:1,:2,:3,:4,:5)";
SEQBindParaStru seq_bind_para;
SEQDBErrorStru out_db_error;

seq_bind_para.length(5);
seq_bind_para[0].dci_type = BIND_STR;
seq_bind_para[0].data_size = 20;
seq_bind_para[1].dci_type = BIND_STR;
seq_bind_para[1].data_size = 64;
seq_bind_para[2].dci_type = BIND_INT;
seq_bind_para[2].data_size = sizeof(int);
seq_bind_para[3].dci_type = BIND_ODT;
seq_bind_para[3].data_size = sizeof(BindDate);
seq_bind_para[4].dci_type = BIND_FLT;
seq_bind_para[4].data_size = sizeof(float);
int row_num = 5000;
char * data_buffer = (char *)malloc(row_num * (20 + 64 + sizeof(int) + sizeof(BindDate) + sizeof(float)));

int switch_buffer = 0;
char tmp_id[20];
sprintf(tmp_id,"%ld",pthread_self());
string id = string(tmp_id);
string test_name = "test_bind";
int test_age = 19;
time_t now_time = time(0);
tm *ltm = localtime(&now_time);
```

```
BindDate bindDate;
bindDate.DateYYYY = 1900 + ltm->tm_year;
bindDate.DateMM = 1 + ltm->tm_mon;
bindDate.DateDD = ltm->tm_mday;
bindDate.DateTime.TimeHH = ltm->tm_hour;
bindDate.DateTime.TimeMI = ltm->tm_min;
bindDate.DateTime.TimeSS = ltm->tm_sec;
int ret_code = 0;
float test_float = 2.98;

int i = 0;
string tmp_bind_id;
for (i = 0; i < row_num; ++i) {
   tmp_bind_id = id + to_string(i);
   memcpy(data_buffer + switch_buffer, tmp_bind_id.c_str(), 20);
   switch_buffer += 20;
   memcpy(data_buffer + switch_buffer, test_name.c_str(), 64);
   switch_buffer += 64;
   memcpy(data_buffer + switch_buffer, &test_age, sizeof(int));
   switch_buffer += sizeof(int);
   memcpy(data_buffer + switch_buffer, &bindDate, sizeof(BindDate));
   switch_buffer += sizeof(BindDate);
   memcpy(data_buffer + switch_buffer, &test_float, sizeof(float));
   switch_buffer += sizeof(float);
}
ret_code = sql_sp_client.ExecuteBind(bind_str.c_str(), seq_bind_para, data_buffer, NULL,
row_num, out_db_error);
if (ret_code >= 0)
{
   cout << " ExecuteBind 执行成功" << endl;
}else{
   cout << " ExecuteBind 执行失败" << endl;
}
free(data_buffer);
delete sql_sp_client;
sql_sp_client = NULL;
```

A.7.6 异步数据提交接口

A.7.6.1 接口原型：short SqlCommit(int sql_num, const StrSeq& seq_sql_str,);

A.7.6.2 接口说明：异步数据提交接口，传入一条或者多条提交或者修改类型的 sql 语句。接口调用成功返回 1，仅表示数据异步提交服务接收到 sql 语句，不代表商用库提交成功；接口调用失败则返回-1，表示数据异步提交服务未接收到 sql 语句。

A.7.6.3 参数列表见表 A.115：

表 A.115 SqlCommit 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
int sql_num	int	In	需执行的修改 SQL 语句总数	是
const StrSeq& seq_sql_str	StrSeq	In	记录多条修改 SQL 语句的结构	是
返回值	short	Out	1：服务端调用成功	是

			-1：服务端调用失败	

A.7.6.4 参数说明：

```
// 记录多条修改 SQL 语句的结构
typedef MLang::VECTOR<MLang::STRING> StrSeq;
```

A.7.6.5 接口示例：

```
using namespace JK_API;
CJKCommitClient *commit_client = new CJKCommitClient ();
StrSeq  str_seq;
String sql_str1 = "update test1111 set name = 'aaa' where id = 281193501733945345";
String sql_str1 = "update test1111 set name = 'bbb' where id = 281193501733945346";
str_seq.push_back(sql_str1.c_str());
str_seq.push_back(sql_str2.c_str());

retcode = commit_client->SqlCommit(2, seq_sql_str);
cout << "SqlCommit retcode = " << retcode << endl;
if (retcode == 1)
{
  cout << "服务端调用成功" << endl;
}
else{
      cout << "服务端调用失败" << endl;
}

delete commit_client;
commit_client = NULL;;
```

A.7.7 异步批量数据提交接口

A.7.7.1 接口原型：short SqlCommitWithBind(TCommitBindStru CommitBindStru);

A.7.7.2 接口说明：异步批量数据提交接口，通过绑定变量方式传入 sql 语句和数据，接口调用完成返回 1，否则返回-1。接口调用成功返回 1，仅表示数据异步提交服务接收到绑定变量的入库信息，不代表商用库提交成功；接口调用失败则返回-1，表示数据异步提交服务未接收到绑定变量的入库信息。

A.7.7.3 参数列表见表 A.116：

表 A.116 SqlCommitWithBind 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
CommitBindStru	TCommitBindStru	In	需执行的绑定变量语句结构	是
返回值	short	Out	1：提交成功 -1：提交失败	是

A.7.7.4 参数说明：

```
// TCommitBindStru 结构说明
struct UDataValue
{
private:
      union
      {
            char*       c_string;
            unsigned char  c_uchar;
            short       c_short;
            int    c_int;
```

```
            MLang::Long c_time;
            float c_float;
            double     c_double;
            TKeyID   c_keyid;
            CharSeq*c_binary;
            CharSeq*c_text;
            CharSeq*c_image;
            TAppKeyID    c_appkeyid;
            TAppID   c_appid;
            char c_default;
            unsigned int    c_uint;
            MLang::Long c_long;
        }__u_val;
        bool __u_init;
        unsigned short __u_index;
public:
        UDataValue();
        ~UDataValue();
        UDataValue(const UDataValue&);
        UDataValue&operator=(const UDataValue&);
        short _d()const;
        void __clear();
        void __write(MLang::OutputStream&__os)const;
        void __read(MLang::InputStream&__is);
        void c_string(const char*);
        void c_string(const MLang::STRING&);
        const char*c_string()const;
        void c_uchar(const unsigned char);
        const unsigned charc_uchar()const;
        void c_short(const short);
        const short      c_short()const;
        void c_int(const int);
        const int  c_int()const;
        void c_time(const MLang::Long);
        const MLang::Longc_time()const;
        void c_float(const float);
        const float       c_float()const;
        void c_double(const double);
        const double    c_double()const;
        void c_keyid(const TKeyID);
        const TKeyID c_keyid()const;
        void c_binary(const CharSeq&);
        const CharSeq&c_binary()const;
        CharSeq&c_binary();
        void c_text(const CharSeq&);
        const CharSeq&c_text()const;
        CharSeq&c_text();
        void c_image(const CharSeq&);
        const CharSeq&c_image()const;
        CharSeq&c_image();
        void c_appkeyid(const TAppKeyID&);
        const       TAppKeyID&c_appkeyid()const;
        TAppKeyID&c_appkeyid();
        void c_appid(const TAppID&);
        const       TAppID&c_appid()const;
```

```
        TAppID&c_appid();
        void c_default(const char);
        const char       c_default()const;
        void c_uint(const unsigned int);
        const unsigned int   c_uint()const;
        void c_long(const MLang::Long);
        const MLang::Longc_long()const;
};
struct TBindDataValue
{
        UDataValue    data_value;
        unsigned short is_null;
        void __write(MLang::OutputStream&__os)const;
        void __read(MLang::InputStream&__is);
};
typedef MLang::VECTOR<TBindDataValue> SEQBindDataValue;
struct TBindCommitParaStru
{
        MLang::STRING    col_name;
        unsigned short is_where_condition;
        unsigned short dci_type;
        unsigned int    data_size;
        TBindCommitParaStru();
        TBindCommitParaStru(const TBindCommitParaStru&);
        TBindCommitParaStru&operator=(const TBindCommitParaStru&);
        void __write(MLang::OutputStream&__os)const;
        void __read(MLang::InputStream&__is);
};
typedef MLang::VECTOR<TBindCommitParaStru> SEQBindCommitParaStru;
struct TCommitBindStru
{
        MLang::STRING    table_name;
        unsigned int    row_num;
        unsigned int    col_num;
        unsigned int    op_type;
        SEQBindCommitParaStru      seq_bind_para;
        SEQBindDataValueseq_bind_value;
        TCommitBindStru();
        TCommitBindStru(const TCommitBindStru&);
        TCommitBindStru&operator=(const TCommitBindStru&);
        void __write(MLang::OutputStream&__os)const;
        void __read(MLang::InputStream&__is);
};
```

A.7.6.5 接口示例：

```
using namespace JK_API;
CJKCommitClient *commit_client = new CJKCommitClient ();
TCommitBindStru CommitBindStru;
CommitBindStru.table_name = "TEST111111" ;
CommitBindStru.row_num = 100;
CommitBindStru.col_num = 2 ;
CommitBindStru.op_type = OP_BIND_INSERT ;

CommitBindStru.seq_bind_para.length(CommitBindStru.col_num);
CommitBindStru.seq_bind_para[0].col_name = "ID";
CommitBindStru.seq_bind_para[0].data_size = 50;
```

```
CommitBindStru.seq_bind_para[0].dci_type = BIND_COMMIT_DATATYPE_STR ;
CommitBindStru.seq_bind_para[0].is_where_condition = 0;
CommitBindStru.seq_bind_para[1].col_name = "NAME";
CommitBindStru.seq_bind_para[1].data_size = 50;
CommitBindStru.seq_bind_para[1].dci_type = BIND_COMMIT_DATATYPE_STR ;
CommitBindStru.seq_bind_para[1].is_where_condition = 0;

CommitBindStru.seq_bind_value.length(2*100);

for(int i = 0 ; i < 100 ; i ++)
{
    char buf1[64] = {0};
    char buf2[64] = {0};
    int seq = row_num * k + i;
    sprintf(buf1, "%s%010d", argv[2], seq);
    sprintf(buf2, "%s%010d", argv[3], seq);
    CommitBindStru.seq_bind_value[col_num*i].is_null = 0;
    CommitBindStru.seq_bind_value[col_num*i].data_value.c_string(buf1);
    CommitBindStru.seq_bind_value[col_num*i+1].is_null = 0;
    CommitBindStru.seq_bind_value[col_num*i+1].data_value.c_string(buf2);
}
retcode = commit_client->SqlCommitWithBind(CommitBindStru);
cout  << " SqlCommitWithBind retcode = " << retcode << endl;
delete commit_client;
commit_client = NULL;
```

A.8 消息总线接口

A.8.1 总体要求

消息总线作为一个重要的通信中间件模块，为系统不同进程之间、不同机器之间提供信息与数据的传输服务。通过屏蔽底层平台之间的异构性，简化了不同应用之间的消息传递与数据交互。消息总线通过对外接口：消息初始化、消息订阅、消息接收、消息发送、消息退出接口来提供消息收发功能。接口描述参见示例。接口头文件及动态库名称见基础说明 2.8。

A.8.2 消息注册

A.8.2.1 接口原型：int jkmessage_invocation::messageInit(char*context_name,char*app_name,char*proc_name)。

A.8.2.2 接口说明：使用消息总线的所有进程都需要先初始化。初始化主要完成对文件映射等初始化工作，函数返回值是系统中全局唯一的值，进程注册消息的键值。

A.8.2.3 参数列表见表 A.117：

表 A.117 messageInit 原语参数列表

接口参数	类型	输入输出	参数说明	是否必填
context_name	char *	In	注册的态名	是
app_name	char *	In	注册的应用名	是
proc_name	char *	In	注册的进程名	是
返回值	int	Out	>=0：成功; <0：失败。	是

A.8.2.4 调用示例：

```
#include “jk_message_inv.h”
```

```
jkmessage_invocation  msg_bus；
int main(int argc,char**argv)
        {
                int  rtn = msg_bus.messageInit("realtime",“public",“msgsend");//初始化
                if(rtn < 0){printf("Bus init failed!!\n");return -1;}
                return 1；
}
```

A.8.3　消息订阅

A.8.3.1 接口原型：int jkmessage_invocation::messageSubscribe(short set_id, char *context_name=NULL)。

A.8.3.2 接口说明：同一类型的消息称为一个通道，订阅某一通道的消息后，发送到该通道的所有消息都会给此进程分发一份。调用该接口时会将本进程的进程 ID 号加入到通道对应进程组中。

A.8.3.3 参数列表见表 A.118：

表 A.118 messageSubscribe 原语参数列表

接口参数	类型	输入输出	参数说明	是否必填
set_id	short	In	订阅的通道名	是
context_name	char *	In	订阅的态名	否
返回值	int	Out	>=0：成功; <0：失败。	是

A.8.3.4 调用示例：

```
        #include “jk_message_inv.h”
        jkmessage_invocation  msg_bus；
int main(int argc,char**argv)
        {
                int  rtn =
                msg_bus.messageInit("realtime",“public",“msgsend");//初始化
                if(rtn < 0){printf("Bus init failed!!\n");return -1;}
                rtn = msg_bus.messageSubscribe(6, "realtime");
                if(rtn>0){
                  …….
                }else{
                ……
                }
                return 1；
        }
```

A.8.4　取消订阅

A.8.4.1　接口原型：int jkmessage_invocation::messageUnSubscribe(short set_id, char *context_name=NULL)。

A.8.4.2　接口说明：订阅某一通道的消息后，因需要取消订阅该通道的消息。将本进程 id 从通道进程组中抹去。

A.8.4.3　参数列表见表 A.119：

表 A.119 messageUnSubscribe 原语参数列表

接口参数	类型	输入输出	参数说明	是否必填
set_id	short	In	订阅的通道名	是
context_name	char *	In	订阅的态名	否

返回值	int	Out	>=0：成功; <0：失败。	是

A.8.4.4 调用示例：

```
#include “jk_message_inv.h”
jkmessage_invocation  msg_bus；
int main(int argc,char**argv)
{
int  rtn =
msg_bus.messageInit("realtime",“public",“msgsend");//初始化
if(rtn < 0){printf("Bus init failed!!\n");return -1;}
rtn = msg_bus.messageSubscribe(6, "realtime");
if(rtn>0){
  …….
}else{
……
}
rtn = msg_bus.messageUnSubscribe(6, "realtime");//取消订阅
return 1；
}
```

A.8.5 消息接收

A.8.5.1 接口原型：int jkmessage_invocation::messageReceive(JKMessage *messageP,JKMsg_source *msg_src_p=NULL, int sync=1)。

A.8.5.2 接口说明：调用该接口可以发送指定通道的消息到消息总线；接收消息接口分异步接口和同步接口两种：异步接受消息时，不管有无消息接口立即返回，返回值表示接收消息的长度；同步接收时，无消息时该调用会被阻塞函数返回值，返回正确接收的消息内容长度。当使用异步接口时，要用返回值作判断是否接收到消息。

A.8.5.3 参数列表见表 A.120：

表 A.120 messageReceive 原语参数列表

接口参数	类型	输入输出	参数说明	是否必填
messageP	JKMessage *	Out	接收消息内容	是
msg_src_p	JKMsg_source *	Out	接收消息源	否
sync	Int	In	接收方式： 0/1:异步接收/同步接收	否
返回值	int	Out	>=0：成功; <0：失败。	是

A.8.5.4 调用示例：

```
#include “jk_message_inv.h”
jkmessage_invocation  msg_bus；
int main(int argc,char**argv)
{
int  rtn =
msg_bus.messageInit("realtime",“public",“msgsend");//初始化
if(rtn < 0){printf("Bus init failed!!\n");return 0;}
rtn = msg_bus.messageSubscribe(6, "realtime");
if(rtn>0){
  …….
}else{
……
```

```
}
JKMessage        msg;
JKMsg_source    JKMsg_source;
//以异步方式接收消息
while(true)
{
      rtn =msg_bus.messageReceive(&msg,&JKMsg_source,0);
      if(rtn <= 0)
   {
         //Do something
     sleep(1);
      continue;
        }
     //Do something else
      }
      //同步方式接收
      while(true)
      {
      rtn =msg_bus.messageReceive(&msg,&JKMsg_source,1);
      //Do something
      }
return 1;
}
```

A.8.6 消息发送

A.8.6.1 接口原型：int jkmessage_invocation::messageSend(JKMessage*messageP,int messageLength, JKMsg_destination *msg_dst_p=NULL)。

A.8.6.2 接口说明：调用该接口可以发送指定通道的消息到消息总线。

A.8.6.3 参数列表见表 A.121：

表 A.121 messageSend 原语参数列表

接口参数	类型	输入输出	参数说明	是否必填
messageP	JKMessage *	IN	待发送的消息体	是
messageLength	int	IN	消息体中数据的长度	是
msg_dst_p	JKMsg_destination *	IN	发送目标地址: p==NULL 发送通道报文，p!=NULL 发送点对点报文	否
返回值	int	Out	>=0：成功; <0：失败。	是

A.8.6.4 调用示例：

```
#include “jk_message_inv.h”
jkmessage_invocation  msg_bus;
int main(int argc,char**argv)
{
int  rtn =
msg_bus.messageInit("realtime",“public",“msgsend");//初始化
if(rtn < 0){printf("Bus init failed!!\n");return -1;}
//发送 UDP 报文到某一通道
JKMessage msg;
msg.header.serv=6; //通道号，目前介于 0~255
msg.header.event=10; //事件类型，目前介于 0~1299 之间
```

```
strcpy(msg.Msg_buf,"udp message send test!!\n");
int buf_len = strlen(msg.Msg_buf)+1;//
rtn =msg_bus.messageSend(&msg,buf_len,NULL);
if(rtn>0)
printf("Send udp message successfully!!\n");
else
printf("Send udp message failed!!\n");
//发送 TCP 报文
JKMsg_destination tcp_dst;
strcpy(tcp_dst.host_name,"sca1-1"); //目标主机
tcp_dst.sock = 0; //初始化
tcp_dst.status = 1; //发送过程中链接一直被保持
rtn = msg_bus.messageSend(&msg,buf_len,&tcp_dst);
if(rtn>0)
printf("Send tcp message successfully!!\n");
else
printf("Send tcp message failed!!\n");
return 1;
}
```

A.8.7 消息退出

A.8.7.1 接口原型：int jkmessage_invocation::messageExit (int proc_key=-1)。

A.8.7.2 接口说明：退出消息总线，清理订阅的所有通道，设置状态等。

A.8.7.3 参数列表见表 A.122：

表 A.122 messageExit 原语参数列表

接口参数	类型	输入输出	参数说明	是否必填
proc_key	int	Out	注册消息的进程键值	否
返回值	int	Out	>=0：成功; <0：失败	是

A.8.7.4 调用示例：

```
jkmessage_invocation  msg_bus;
int  rtn = msg_bus.messageInit("realtime",“public",“msgsend");//初始化
msg_bus.messageExit();
```

A.8.8 数据结构定义

A.8.8.1 消息结构体（JKMessage）

消息结构体JKMessage用于定义消息的组成，由消息头和消息体构成，其数据结构声明如下：

```
{
JKMessageHeader    messageheaer;                    //消息头
Char            Msg_buf [JK_MAX_MSGBUF_LEN];        //消息体
  //消息体
  } JKMessage
```

其中消息体长度上限MAX_MSGBUF_LEN为32767Byte。

A.8.8.2 消息头（JKMessageHeader）

消息头JKMessageHeader的数据结构声明如下：

```
typedef struct
{
    short        len;       //消息体长度
    short        serv;      //通道号 ID，应用填写
```

```
    short       seq;      //消息序号
    short       event;    //事件号 ID，应用填写
    unsigned char   domain;   //消息域号
    unsigned char   ctxt;     //消息态号
    short       stid;     //消息源信息
    short       dtid;     //消息目的信息
    unsigned char   ver_coding; //消息类型
    unsigned char   remain;   //保留字段
}JKMessageHeader;
```

A.8.8.3 消息目的地址信息

消息目的地址信息（JKMsg_destination）用于定义消息发送方式、目的地址等信息，数据结构声明如下：

```
typedef struct
{
  char   host_name[JK_MAX_STRING_LEN_BUS];
  char   context_name[JK_MAX_STRING_LEN_BUS];
  char   app_name[JK_MAX_STRING_LEN_BUS];
  char   proc_name[JK_MAX_STRING_LEN_BUS];
  int    addr;
  int    port;
  int    status;
  int    sock;
  int    ctxt;
  int    serv;
  int    event;
}JKMsg_destination;
```

A.8.8.4 消息源地址信息

消息源地址信息（JKMsg_source）用于定义消息源地址信息，数据结构声明如下：

```
{
Char      host_name [JK_MAX_STRING_LEN_BUS];   //节点名
Char      context_name[JK_MAX_STRING_LEN_BUS];  //态名
Char      app_name[JK_MAX_STRING_LEN_BUS];    //应用名
Char      proc_name[JK_MAX_STRING_LEN_BUS];    //进程名
} JKMsg_source
```

A.9 服务总线接口

A.9.1 服务端接口

应满足以下要求：

a) 服务端初始化：

1) 接口原型：int JkServiceServerInit(JK_ServiceInfo serviceinfo, int mode)
2) 接口说明：该接口用于请求/响应模型服务端的初始化等
3) 参数列表见表 A.123：

表 A.123 JkServiceServerInit 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
serviceinfo	JK_ServiceInfo	In	服务的连接信息（如地址、端口号等）	是
mode	int	In	服务端运行方式，默认（mode=JK_DISPATCH）	是
返回值	int	Out	=1：成功; <0：失败	—

b) 服务分发-单次响应方式：

1) 接口原型：typedef void* (*JK_Func)(char* requestBuffer, int requestlen, char** responseBuffer, int* responselen);int JkServiceDispatch (JK_ServiceInfo serviceinfo, int flag, JK_Func func)
2) 接口说明：该接口完成请求报文的接收和任务的分发，有多线程和单线程两种方式。对于多线程方式，接口为每个请求启动一个线程，接受请求报文，然后调用 func 回调函数，要求 func 必须是线程安全的，函数内部对于关键区域必须互斥。对于单线程方式，不再启动线程。
3) 参数列表见表 A.124：

表 A.124 JkServiceDispatch 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
serviceinfo	JK_ServiceInfo	In	服务的连接信息（如地址、端口号等）	是
flag	int	In	服务端运行方式（多线程：2，单线程：0）	是
func	JK_Func	In	回调函数	是
requestBuffer	char*	In	要处理的请求报文	是
requestlen	int	In	请求报文长度	是
responseBuffer	char**	Out	响应报文	是
responselen	int*	Out	响应报文长度	是
返回值	int	Out	=1：成功; <0：失败	—
注意：回调函数 func 中的 responseBuffer 指向的响应报文所需内存由回调函数使用 malloc 分配，服务总线调用回调函数后将释放该内存。				

c) 订阅内容注册函数：
   1) 接口原型：typedef int JK_Determine(const char *requestBuffer, const int requestlen);
   int JkServicePublishRegister(JK_ServiceInfo serviceinfo, JK_Determine* determine_func = NULL, const int handle_num = 1);
   2) 接口说明：注册订阅/发布结果类型
   3) 参数列表见表 A.125：

表 A.125 JkServicePublishRegister 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
serviceinfo	JK_ServiceInfo	In	服务的连接信息（如地址、端口号等）	是
determine_func	Determine*	In	订阅内容与发布句柄分发函数。 返回值为发布句柄	是
handle_num	int	In	发布句柄个数，根据服务端发布的主题个数来定	是
返回值	int	Out	=0：成功; <0：失败	—

d) 订阅结果发布：
   1) 接口原型：int JkServicePublish(const char *responseBuffer, const int responselen, const int handle_index = 0);
   2) 接口说明：订阅结果发布函数
   3) 参数列表见表 A.126：

表 A.126 JkServicePublish 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
responseBuffer	char**	In	发布结果报文	是
responselen	int*	In	发布结果报文长度	是
handle_index	int	In	发布结果句柄，为 JkServicePublishRegister 中注册的 JK_Determine 函数中返回的句柄， 必须小于 handle_num，如果只提供一种类型的结果发布，该参数不填写	是
返回值	int	Out	=0：成功 =-1：失败 =-2：无订阅	—

e) 服务注册函数：
   1) 接口原型：int JkServiceRegisterInit(const char *pname, const char *pcname, const char

*paname, int servid, port, int balance = 0)

2) 接口说明：服务端向服务管理中心注册服务基础信息，便于服务定位、查询和管理。

3) 参数列表见表 A.127：

表 A.127 JkServiceRegisterInit 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
pname	const char *	In	服务名	是
pcname	const char *	In	服务所属的态名	是
paname	const char *	In	服务所属的应用名	是
servid	int	In	服务 id	是
port	int	In	端口号	是
balance	int	In	负载均衡标志， 1 负载均衡，0 为主备均衡	否
返回值	int	Out	=0：成功; <0：失败	—

A.9.2 局域客户端接口

应满足以下要求：

a） 客户端定位服务函数：

1) 接口原型：int JkServiceLocate(const char * cxt_name, const char * app_name, const char * servname, JK_ServiceInfo *psi, const int balance)

2) 接口说明：该接口用于对要访问的服务进行定位

3) 参数列表见表 A.128：

表 A.128 JkServiceLocate 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
cxt_name	const char *	In	服务所属的态名	是
app_name	const char *	In	服务所属的应用名	是
servname	const char *	In	服务名	是
psi	JK_ServiceInfo *	In	返回服务定位结果(服务 ip 地址和端口号)	是
balance	int	In	1 负载均衡，0 为主备均衡，默认为 0	否
返回值	int	Out	=1：成功; <0：失败	—

b） 同步请求服务函数：

1) 接口原型：int JkServiceRequestSync (JK_ServiceInfo serviceinfo, const char *requestBuffer, int requestlen, time_t timeout, char ** responseBuff, int *responselen, JK_Handle *requesthandle);

2) 接口说明：客户端提交服务请求，阻塞等待结果返回。
3) 参数列表见表 A.129：

表 A.129 JkServiceRequestSync 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
serviceinfo	JK_ServiceInfo	In	服务连接信息， 由 JkServiceLocate 返回	是
requestBuffer	const char *	In	请求报文	是
requestlen	int	In	请求报文长度	是
timeout	time_t	In	超时时间	是
responseBuff	char **	Out	响应报文	—
responselen	int *	Out	响应报文长度	—
requesthandle	JK_Handle *	Out	请求句柄，用于 JkServiceRequestFree 和 JkServiceHandleFree 及链路复用。由原语返回	—
返回值	int	Out	=0：成功; <0：失败	—
注：第一次使用时 requesthandle 初始化为 JK_HANDLE_INIT，以后直接使用即可，复用同一条链路；在返回出错后，建议释放请求句柄，重新初始化 JK_HANDLE_INIT 为进行重新连接（以下链路复用的接口与此类似）。				

c） 异步请求接口：
1) 接口原型：int JkServiceRequestAsync (JK_ServiceInfo serviceinfo, const char *requestBuffer, int requestlen, time_t timeout, JK_Handle *requesthandle);
2) 接口说明：客户端提交基于 S 语言的服务请求，结果由 JkServiceReqSyncTest（）函数返回。不需要返回结果则调用服务请求资源释放函数即可。
3) 参数列表见表 A.130：

表 A.130 JkServiceRequestAsync 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
serviceinfo	JK_ServiceInfo	In	服务连接信息， 由服务管理返回	是
requestBuffer	const char *	In	请求报文	是
requestlen	int	In	请求报文长度	是
timeout	time_t	In	超时时间	是
requesthandle	JK_Handle *	Out	请求句柄，用于 JkServiceRequestFree 和 JkServiceHandleFree 及链路复用。由原语返回	—
返回值	int	Out	=0：成功 <0：失败	—

d） 异步请求结果：
1) 接口原型：int JkServiceReqSyncTest (JK_Handle requesthandle, char ** responseBuff, int buflen);
2) 接口说明：读取异步请求的结果，如果请求任务未完成，则阻塞等待结果返回；如果已完成，但是失败，则返回失败。

3) 参数列表见表 A.131：

表 A.131 JkServiceReqSyncTest 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
requesthandle	JK_Handle	In	请求句柄，由 JkServiceRequestAsync 返回	是
responseBuff	char **	Out	响应报文	—
responselen	int *	Out	响应报文长度	—
返回值	int	Out	<0：失败 =1：成功	—
注释：每次调用异步请求结果接口后均需调用 JkServiceRequestFree 资源释放接口释放请求服务函数申请的资源。				

e） 服务请求资源释放函数：

1) 接口原型：int JkServiceRequestFree (JK_Handle requesthandle)
2) 接口说明：释放请求服务函数申请的资源。
3) 参数列表见表 A.132：

表 A.132 JkServiceRequestFree 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
requesthandle	JK_Handle	In	请求句柄	是
返回值	int	Out	=0：成功; <0：失败	—

f） 句柄释放函数：

1) 接口原型：int JkServiceHandleFree (JK_Handle requesthandle)
2) 接口说明：释放请求句柄
3) 参数列表见表 A.133：

表 A.133 JkServiceHandleFree 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
requesthandle	JK_Handle	In	请求句柄	是
返回值	int	Out	=0：成功; <0：失败	—

g） 服务订阅函数：

1) 接口原型：Typedef int JK_Function (char * responseBuffer, int buflen);int JkServiceSubscribe(JK_ServiceInfo serviceinfo, int flag, const char *requestBuffer, int requestlen, JK_Function* func, JK_Handle *handle)
2) 接口说明：该接口完成服务的订阅。首先按照 flag 制定的方式订阅服务，然后等待订阅确认，如果确认失败，则直接结束；如果确认成功，则创建线程，等待订阅消息的返回，每收到一个回送来的结果，就调用回调函数 func 执行数据处理。

4) 参数列表见表 A.134：

表 A.134 JkServiceSubscribe 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
serviceinfo	JK_ServiceInfo	In	服务连接信息， 由 JkServiceLocate 返回	是
requestBuffer	const char *	In	请求报文	是
requestlen	int	In	请求报文长度	是
flag	int	In	同步，默认为 0	否
handle	JK_Handle *	Out	请求句柄，用于 JkServiceRequestFree 和 JkServiceHandleFree 及链路复用。由原语返回	—
responseBuffer	char *	Out	响应报文	—
buflen	int	Out	响应报文长度	—
返回值	int	Out	=0：成功; <0：失败	—
注：订阅接口每个 handle 只支持订阅一个主题。				

h） 订阅取消函数：

1) 接口原型：int JkServiceUnSubscribe(JK_Handle handle)
2) 接口说明：该接口取消对应的内容。
3) 参数列表见表 A.135：

表 A.135 JkServiceUnSubscribe 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
handle	JK_Handle	In	请求句柄	是
返回值	int	Out	=0：成功; <0：失败	—

A.9.3 数据结构定义

应满足以下要求：

a） 服务信息数据结构：

1) 数据结构原型：

```
typedef  struct {
int addr;
int port;
int serv;
char servname[64];
char state;
char keep_alive;      //to set key in ServiceHeader
short reserved;
```

}JK_ServiceInfo;

2） 数据结构说明：JK_ServiceInfo 用于客户端和服务端在调用服务总线接口时输入基本的服务访问信息。注：一般情况下客户端发送请求接口的 JK_ServiceInfo 信息由服务定位接口返回。服务端接口 JK_ServiceInfo 信息只需要填充监听端口即可。

3） 参数列表见表 A.136：

表 A.136 JK_ServiceInfo 数据结构成员列表

成员列表	类型	输入/输出	成员说明	是否必填
addr	int	In	服务端地址	是
port	int	In	服务端端口	是
serv	int	In	服务 ID	否
servname	char	In	服务名	是
state	char	In	服务状态	否
keep_alive	char	In	0：请求/响应模型 1：订阅/发布模型	是
reserved	short	In	预留字段	否

b） 域信息数据结构：

1） 数据结构原型：

```
typedef struct JK_DomainInfo
{
    char    domain[JK_MAXLEN];      // 域名, 如：华北.河北(domain 值与域配置文件中本地信息相同表示请求本地主机, 否则为请求远程主机)
    int     appno;                  // 应用号
    int     context;                // 态号
    char    servicename[JK_MAXLEN]; // 服务名
    tSecLabel   pslabel;            // 标签
}JK_DomainInfo;
```

注：广域应用程序需要填充要访问远端机构的域名、应用号、态号和服务名字段，角色标签字段不需要填充。

2） 数据结构说明：JK_DomainInfo 用于广域客户端调用广域服务请求接口时输入基本的服务访问信息。

3） 参数列表见表 A.137：

表 A.137 JK_DomainInfo 数据结构成员列表

成员列表	类型	输入/输出	成员说明	是否必填
domain	int	In	域名	是
appno	int	In	应用号	是
context	int	In	态号	是

servicename	char	In	服务名	是
pslabel	JK_tSecLabel	In	安全相关数据结构，参考安全相关文档	否

A.9.4 服务总线交互协议

协议内容包含两个部分：消息头和报文体。

消息头	报文体
JK_ServiceHeader	Char[]

```
typedef struct
{
    int     len;            //报文长度
    short       serv;
    short       seq;
    short       errinfo;
    short       stid;
    short       dtid;
    short       key;        // 模式（请求响应：0  订阅发布：1）
}JK_ServiceHeader;
```

A.9.5 广域客户端接口

应满足以下要求：

a) 远程同步请求接口：

1) 接口原型：

```
int JkRemoteRequestSync(JK_DomainInfo       *domaininfo,
                        const char          *requestBuffer,
                        int                 requestlen,
                        time_t              timeout,
                        char            **responseBuff,
                        int                 *responselen,
                        JK_Handle               *handle,
                        int                 resendFlag = 1,
                        const char          *IP = NULL);
```

2) 接口说明：该接口用于远程服务请求（单次请求单次响应），阻塞等待结果返回，可以设置等待超时时间。

3) 参数列表见表 A.138：

表 A.138 JkRemoteRequestSync 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
domaininfo	JK_DomainInfo *	In	域信息	是
requestBuffer	char *	In	请求报文	是
requestlen	int	In	请求报文长度	是
timeout	time_t	In	超时时间，默认 60 秒	否

responseBuff	char **	Out	响应报文	—
responselen	int *	Out	响应报文长度	—
handle	JK_Handle *	Out	请求句柄	—
resendFlag	int	In	重传标志	否
IP	const char *	In	用于连接指定 IP 地址代理	否
返回值	int	Out	>0：成功 <=0 失败	—
注： 1) 第一次使用时 handle 初始化为 JK_HANDLE_INIT，以后直接使用即可，复用同一条链路； 2) 接口调用失败不需要释放任何资源（接口内部已自动释放）。				

b) 远程请求服务资源释放接口:
   1) 接口原型：int JkRemoteRequestFree(JK_Handle requesthandle);
   2) 接口说明：释放请求服务函数申请的资源，不调用此接口会导致内存泄漏。
   3) 参数列表见表 A.139：

表 A.139 JkRemoteRequestSync 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
requesthandle	JK_Handle	In	请求句柄	是
返回值	int	Out	=0：成功; <0：失败	—

c) 远程请求句柄释放接口：
   1) 接口原型：int JkRemoteHandleFree (JK_Handle requesthandle);
   2) 接口说明：释放请求句柄，关闭连接。
   3) 参数列表见表 A.140：

表 A.140 JkRemoteRequestSync 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
requesthandle	JK_Handle	In	请求句柄	是
返回值	int	Out	=0：成功; <0：失败	—

d) 广域订阅请求接口:
   1) 接口原型：typedef int JK_Function (char *responseBuffer, int buflen);
   int JkRemoteSubscribe(JK_DomainInfo *domaininfo,const char *requestBuffer,int requestlen,JK_Function *func,JK_Handle *handle,int flag = 0 ,const char *IP = NULL);
   2) 接口说明：该接口完成广域服务的订阅。首先按照 flag 制定的方式订阅服务，然后等待订阅确认，如果确认失败，则直接结束；如果确认成功，等待订阅消息的返回，每收到

一个回送来的结果，就调用回调函数 func 执行数据处理。

3) 参数列表见表 A.141：

表 A.141 JkRemoteSubscribe 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
domaininfo	JK_DomainInfo *	In	域信息	是
requestBuffer	Const char *	In	请求报文	是
requestlen	int	In	请求报文长度	是
func	JK_Function*	In	回调函数指针，用于接收服务端推送信息。	是
flag	Int	In	同步，默认为 0	否
handle	JK_Handle *	Out	请求句柄，用于 JkRemoteUnSubscribe，由该接口返回。	—
responseBuffer	char *	Out	响应报文	—
buflen	int	Out	响应报文长度	—
IP	const char *	In	用于连接指定 IP 地址代理，默认为 NULL，一般不需要填写。	否
返回值	Int	Out	>0：成功 <=0 失败	—

e) 广域服务订阅取消接口:

1) 接口原型：int JkRemoteUnSubscribe (JK_Handle requesthandle);

2) 接口说明：取消订阅内容；

3) 参数列表见表 A.142：

表 A.142 JkRremoteUnSubscribe 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
requesthandle	JK_Handle	In	请求句柄	是
返回值	int	Out	=0：成功; <0：失败	—

A.9.6 客户端与服务代理交互协议

a） 访问协议分为请求协议和应答协议。

b） 请求协议格式如下：

服务头部	备用字段长度	备用字段	连接请求长度	服务定位连接请求	应用请求长度	应用服务请求

c） 应答协议如下：

服务头部	服务应答

d） 应满足以下要求：

1) 服务头部：与服务总线中的头部定义相一致。
2) 备用字段长度：定义备用字段的长度 32 字节。
3) 备用字段：备用字段的内容（暂未使用）。
4) 连接请求长度：定义连接请求内容的长度。
5) 服务定位连接请求：定义服务链接请求，格式：SOA://域名/态名/应用名/服务名。
6) 应用请求长度：定义应用请求内容的长度。
7) 应用服务请求：定义应用请求，应用服务请求报文是否加密取决于应用和服务之间的安全规定，由应用进行加密，服务代理不做特殊加密。
8) 服务应答：定义服务应答内容，其长度在服务头部中用 len 字段定义。服务应答报文是否加密取决于应用和服务之间的安全规定，由服务端进行加密，服务代理不做特殊加密。

A.9.7 局域服务访问调用示例

A.9.7.1 局域请求响应模型调用示例如下:

A.9.7.1.1 服务端

```
#include ”jk_services.h”
JK_Func Do(char* requestBuffer, int requestlen, char** responseBuffer, int* responselen)
{
      * responselen = xxxx;
      *responseBuffer = (char *)malloc(* responselen );
            // 填写应答内容
      return (void *)1;

}
int main(int argc, char **argv)
{
      int port = xxxx;// 服务端口号
      JK_ServiceInfo serviceinfo;
      serviceinfo.port = port;
      int servid = xxx;//服务 ID，建议与服务端口保持一致
      int balance = 0;
      JkServiceServerInit(serviceinfo, JK_DISPATCH);
      JkServiceRegisterInit(“dispatch”, “realtime”, “public”, servid ,  port, balance);
   JkServiceDispatch (serviceinfo, 2, Do);
   while(1) { // 防止主进程退出
         sleep(10);
      }
}
```

A.9.7.1.2 客户端

```
#include  "jk_services.h"
int main(int argc, char **argv)
{
      JK_ServiceInfo serviceinfo;
      //定位服务所在的节点
      int ret = JkServiceLocate(ctx_name, app_name, proc_name, &serviceinfo,balance);
      serviceinfo.addr = htonl(serviceinfo.addr);
      JK_Handle handle = JK_HANDLE_INIT;
      char *resp_buffer;
```

```
        int respe_len;
        //同步请求
        ret = JkServiceRequestSync(serviceinfo, request, request_len, 10, &resp_buffer,
        &resp_len, &handle);
        if(0 > ret)  {
            printf("serviceRequestSync error : %s\n",  ret);
        }  else {
            printf("response buff len %d\n", resp_len);
            printf("response buff:%s\n", resp_buffer);
        }
        JkServiceRequestFree(handle);   //释放请求服务函数申请的资源
        JkServiceHandleFree(handle);   //释放请求句柄
        return 0;
    }
```

A.9.7.2 订阅发布模型调用示例如下:

A.9.7.2.1 服务端

```
    JK_Determine determine_func(const char *requestBuffer, const int requestlen)
                                                    // //*requestBuffer 要处理的请求报文
    {
        if (0 == strncmp(requestBuffer, "SE", strlen("SE")))
            return 1;
        else if (0 == strncmp(requestBuffer, "DSA", strlen("DSA")))
            return 2;
         return -1;
    }
    int main(int argc, char **argv)
    {
        JK_ServiceInfo serviceinfo;
        serviceinfo.port = port;
        char  pbuf[1024];
        int buf_len =1024,loop = 0;
        ret = JkServicePublishRegister(serviceinfo,determine_func ,2);
        if (0 > ret) {
            printf("JkServicePublishRegister error\n"); exit(1);
        }
        while (1){
            JkServicePublish(pbuf, buf_len, 1)); //订阅结果发布，1 为 determine_func 函数中
返回的句柄
            sleep(1);
        }
    }
```

A.9.7.2.2 客户端

```
    JK_Function scribeHandle(const char * responseBuffer, int buflen)// * responseBuffer 响应报
文
    {
        // 对应答内容进行显示或者处理
    }
    int main(int argc, char **argv)
    {
        JK_ServiceInfo serviceinfo;
        //定位服务所在的节点
        int ret = JkServiceLocate(ctx_name, app_name, proc_name, &serviceinfo);
        serviceinfo.addr = htonl(serviceinfo.addr);
        JK_Handle handle = JK_HANDLE_INIT;
```

```
        ret = JkServiceSubscribe(serviceinfo, isAsync, request,req_len, scribeHandle, &handle);
        if(0 > ret)  {
            printf(stderr, " serviceSubscribe error : %s\n",  ret);
        }
        else {
            printf("serviceSubscribe ok!\n");
        }
        sleep(10);
        JkServiceUnSubscribe(handle);
        return 0;
    }
```

A.9.8　广域服务访问调用示例

A.9.8.1　广域请求响应模型客户端调用示例如下:

```
#include "jk_remoteCallClient.h"

int main(int argc, char **argv)
{
    if (argc != 5) {
        printf("用法:%s [态号] [应用号] [服务名] [域名]\n", argv[0]);
        printf("%d\n",argc);
        return -1;
    }

    JK_DomainInfo domaininfo;
    domaininfo.context = atoi(argv[1]);
    domaininfo.appno = atoi(argv[2]);
    strcpy(domaininfo.servicename, argv[3]);
    strcpy(domaininfo.domain, argv[4]);

    time_t timeout = 10;
    char *responseBuff = NULL;
    int responselen = 0;
    JK_Handle handle = JK_HANDLE_INIT;

    char requestBuffer[1024] = "hello";
    int requestlen = strlen(requestBuffer);

    int ret = JkRemoteRequestSync(&domaininfo, requestBuffer, requestlen, timeout,
&responseBuff, &responselen, &handle);

    if (0 >= ret) {
        printf("服务总线同步请求原语调用失败!\n");
        return -1;
    }

    printf("+++++ 远程同步请求接口调用成功，ret = %d +++++\n", ret);
    printf("响应报文长度:%d\n", responselen);
    printf("响应报文:%s\n", responseBuff);

    JkRemoteHandleFree(handle);
    JkRemoteUnSubscribe(handle);

    return 0;
```

```
        }
```

A.9.8.2 广域订阅发布模型客户端调用示例如下:

```
JK_Function scribeHandle(const char * responseBuffer, int buflen)
{
  // 对响应内容进行显示或者处理
}
int main(int argc, char **argv)
{
    if (argc != 5) {
            printf("用法:%s [态号] [应用号] [服务名] [域名]\n", argv[0]);
            printf("%d\n",argc);
            return -1;
        }

        JK_DomainInfo domaininfo;
        domaininfo.context = atoi(argv[1]);
        domaininfo.appno = atoi(argv[2]);
        strcpy(domaininfo.servicename, argv[3]);
        strcpy(domaininfo.domain, argv[4]);
    JK_Handle handle = JK_HANDLE_INIT;  //请求句柄
    char requestBuffer[1024] = "xxx";//请求报文
    int requestlen = strlen(requestBuffer);//请求报文长度
    int ret = 0;
    ret = JkRemoteSubscribe(&domaininfo, requestBuffer, requestlen, scribeHandle, &handle, 0);
//该接口完成广域服务的订阅。首先按照 flag 制定的方式订阅服务，然后等待订阅确认，如果确认失败，则直接结束；如果确认成功，等待订阅消息的返回，每收到一个回送来的结果，就调用回调函数 func 执行数据处理。
    if (0 >= ret) {
            printf("服务总线同步请求原语调用失败!\n");
            return -1;
        }
        sleep(100);
  JkRemoteUnSubscribe(handle);// 取消订阅内容。
  printf("success:%d\n", nSucceed);
  return 0;}
```

A.10 公共服务

A.10.1 文件服务接口

A.10.1.1 设置访问本系统文件服务SOCKET连接信息

A.10.1.1.1 平台提供统一的文件处理接口，通过调用这些接口访问平台内集成的文件服务，完成文件和目录的各类操作。JK_API::JKFileAdminstrator 类

A.10.1.1.2 访问文件服务时必须构造一个 JK_API::JKFileAdminstrator 类的对象，由该类的接口负责与文件服务端通信。该类主要提供一系列的接口函数完成文件操作功能。

A.10.1.1.3 接口原型：int setSocket(const char* host, int port);

A.10.1.1.4 接口说明：设置访问本系统文件服务 SOCKET 连接信息，用于连接文件服务。

A.10.1.1.5 参数列表见表 A.143：

表 A.143 setSocket 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
host	const char*	In	文件服务主节点名或 IP	是
port	int	In	文件服务端口（10012）	是
返回值	int	Out	=1 成功 =0 失败	—

A.10.1.1.6 处理过程及返回值说明：设置访问本系统文件服务 SOCKET 连接信息，通过该连接信息连接文件服务，无特殊情况不需要判断返回值。

A.10.1.1.7 调用示例;

```
JK_API::JKFileAdminstrator object;
object.setSocket("jk-scd1", 10012);
```

A.10.1.2 设置访问广域文件服务域名信息

A.10.1.2.1 接口原型：int setDomain(const char *domain, JK_tSecLabel tsr);

A.10.1.2.2 接口说明：设置访问广域文件服务域名信息。

A.10.1.2.3 参数列表见表 A.144：

表 A.144 setDomain 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
domain	const char*	In	广域远端域名	是
tsr	JK_tSecLabel	In	保留参数，暂无意义，定义后传入即可	是
返回值	int	Out	=1 成功 =0 失败	—

A.10.1.2.4 处理过程及返回值说明：设置访问广域文件服务域名信息，通过该域名信息连接远端文件服务，无特殊情况不需要判断返回值。

A.10.1.2.5 调用示例：

```
JK_API::JKFileAdminstrator object;
JK_tSecLabel tsr;
object.setSocket("远端域名", tsr);
```

A.10.1.3 解析返回的报错信息

应满足以下要求：

a） 接口原型：char* show_error();

b） 接口说明：文件服务处理失败时，解析返回的报错信息。

c） 参数列表见表 A.145：

表 A.145 show_error 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
--	--	--	--	—
返回值	char*	Out	报错信息	—

d） 处理过程及返回值说明：仅在文件服务处理失败时调用，解析返回的报错信息；

e） 调用示例：

```
JK_API::JKFileAdminstrator object;
printf("%s\n", object.show_error());
```

A.10.1.4 创建文件

A.10.1.4.1 接口原型：int create_file(const char* file_name, const char* body, int len);

A.10.1.4.2 接口说明：该接口用于客户端创建文件。

A.10.1.4.3 参数列表见表 A.146：

表 A.146 create_file 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
file_name	const char*	In	含相对路径文件名	是
body	const char*	In	文件内容	是
len	int	In	文件内容长度	是

返回值	int	Out	=1 成功 =0 失败	—

A.10.1.4.4 处理过程及返回值说明：客户端调用该接口访问文件服务，文件服务根据参数创建文件，若 返回 0，解析返回的报错信息。

A.10.1.4.5 调用示例：

```
int main(int argc, char **argv)
{
    if(4 != argc)
    {
        printf("create_file【file_name】【file_content】【IP】\n");
        return -1;
    }
    char *file_name = (char *)argv[1];
    char *file_content = (char*)argv[2];
    char *IP = (char *)argv[3];
    JK_API::JKFileAdminstrator object;
    object.setSocket(IP, 10012);

    int result = object.create_file(file_name, file_content, strlen(file_content));
    if (0 == result)
    {
        cout << "create file failed: " << object.show_error() << endl;
        return -1;
    }
    cout << "create file " << file_name << " ok!" << endl;
    return 0;
}
```

A.10.1.5 保存/修改文件

A.10.1.5.1 接口原型：int save_file(const char* file_name, const char* body, int len, int option, int& version);

A.10.1.5.2 接口说明：该接口用于客户端保存/修改文件。

A.10.1.5.3 参数列表见表 A.147：

表 A.147 save_file 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
file_name	const char*	In	含相对路径文件名	是
body	const char*	In	文件内容	是
option	int	In	保存类型： 0----只保存编辑结果 1----保存编辑结果，生成新版本	是
version	int	OUT	保存文件的版本号	是
返回值	int	Out	=1 成功 =0 失败；	—

A.10.1.5.4 处理过程及返回值说明：客户端调用该接口访问文件服务，文件服务根据参数保存/修改文件，若返回 0，解析返回报错信息。

A.10.1.5.5 调用示例：

```
int main(int argc, char **argv)
{
    if(4 != argc)
    {
```

```
        printf("save_file【file_name】【file_content】【IP】\n");
        return -1;
    }
    char *file_name = (char *)argv[1];
    char *file_content = (char *)argv[2];
    char *IP = (char *)argv[3];
    int version;

    JK_API::JKFileAdminstrator object;
    object.setSocket(IP, 10012);

    int result = object.save_file(file_name, file_content, strlen(file_content), option, version);
    if (0 == result)
    {
        cout << "save file:" << file_name << "  failed:" << object.show_error() << endl;
        return -1;
    }
    printf("save file:%s ok, version %d\n ", file_name, version);
    return 0;
}
```

A.10.1.6 按版本查询文件信息

A.10.1.6.1 接口原型：int list_file(const char* file_name, int version, JKFileInfo& fileinfo);

A.10.1.6.2 接口说明：根据版本号获取文件信息。

A.10.1.6.3 参数列表见表 A.148：

表 A.148 list_file 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
file_name	const char*	In	含相对路径文件名	是
version	int	In	0 表示取得最新版本文件信息 -1 表示取得原始文件信息（不带日期格式的文件） >0 表示按提供的版本号获取文件信息	是
fileinfo	JKFileInfo	OUT	目录树	是
返回值	int	Out	=1 成功 =0 失败	—

A.10.1.6.4 目录树结构如下：

```
class JKFileInfo
{
public:
    char filename[JK_MAX_FILEINFONAME_LENGTH]; //文件名
    char dirname[JK_MAX_FILEINFONAME_LENGTH]; //目录名
    bool is_dir;   //is_dir=true 表示目录，否则表示文件
    std::vector<JKFileInfo>  fileinfo_list; //表示该目录的子目录或下属文件
    int create_date;          //文件创建日期或下属文件
    int version;              //文件版本
    int64_t size;             //文件大小
    bool isLocked;            //是否已经被锁
```

```
        JKLockInfo lockinfo;          //锁信息
};
#define JK_MAX_FILEINFONAME_LENGTH 512
其中锁信息定义如下：
class JKLockInfo
{
public:
        char lockHost[JK_MAX_LOCKUSR_LENGTH];  //执行锁定操作的主机名
        char lockUser[JK_MAX_LOCKUSR_LENGTH];  //执行锁定操作的用户名
        int  lockTime; //执行锁定操作的时刻
};
#define  JK_MAX_LOCKUSR_LENGTH  512
```

A.10.1.6.5 处理过程及返回值说明：客户端调用该接口访问文件服务，根据版本号获取并返回文件信息，若返回 0，解析返回报错信息。

A.10.1.6.6 调用示例：

```
int main(int argc, char **argv)
{
        if(4 != argc)
        {
                printf("list_file【file_name】【IP】\n");
                return -1;
        }
        char *file_name = (char *)argv[1];
        char *IP = (char *)argv[2];
        JK_API::JKFileInfo fileinfo;

        JK_API::JKFileAdminstrator object;
        object.setSocket(IP, 10012);

        int result = object. list_file (file_name, -1, fileinfo);
        if (0 == result)
        {
                cout << "list file:" << file_name << "  failed:" << object.show_error() << endl;
                return -1;
        }
        printf("list file:dir %s, file_name %sn ", fileinfo.dirname, fileinfo.filename);
        return 0;
}
```

A.10.1.7 根据版本号获取文件内容

A.10.1.7.1 接口原型：int get_file (const char* file_name, int version, char** body, int&len, int transfer_type=0);

A.10.1.7.2 接口说明：根据版本号获取文件内容。

A.10.1.7.3 参数列表见表 A.149：

表 A.149 get_file 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
file_name	const char*	In	含相对路径文件名	是
version	int	In	-1：查询原文件内容 0：查询原文件最新版本文件的内容 >0：按提供的版本号获取文件内容	是
body	char**	OUT	文件内容	是

len	int	OUT	文件内容长度	是
transfer_type	int	In	文件的传输类型 可忽略填写	否
返回值	int	Out	=1 成功 =0 失败	—

A.10.1.7.4 处理过程及返回值说明：客户端调用该接口访问文件服务，根据版本号获取并返回文件内容，若返回 0，解析返回报错信息。

A.10.1.7.5 调用示例：

```
int main(int argc, char **argv)
{
    if(3 != argc)
    {
        printf("get_file【file_name】【IP】\n");
        return -1;
    }
    char *file_name = (char *)argv[1];
    char *IP = (char *)argv[2];
    JK_API::JKFileAdminstrator object;
    object.setSocket(IP, 10012);
    char* p = NULL;
    int len = 0;

    int result = object.get_file(file_name, -1, &p, len);
    if (result == 0)
    {
        cout << "get file failed: " << object.show_error() << endl;
        return -1;
    }
    printf("%.*s\n", len, p);
    return 0;
}
```

A.10.1.8 按版本号删除文件

A.10.1.8.1 接口原型：int delete_file(const char* file_name，int version);

A.10.1.8.2 接口说明：根据版本号删除文件。

A.10.1.8.3 参数列表见表 A.150：

表 A.150 delete_file 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
file_name	const char*	In	含相对路径文件名	是
version	int	In	-1：删除原始文件 0 ：删除最新版本文件 >0：删除指定版本文件	是
返回值	int	Out	=1 成功 =0 失败；	—

A.10.1.8.4 处理过程及返回值说明：客户端调用该接口访问文件服务，根据版本号删除文件，若返回 0，解析返回的报错信息。

A.10.1.8.5 调用示例：

```
int main(int argc, char **argv)
{
    if(4 != argc)
    {
```

```
        printf("delete_file【file_name】【version】【IP】\n");
        return -1;
    }
    char *file_name = (char *)argv[1];
    int version = atoi((char*)argv[2]);
    char *IP = (char *)argv[3];

    JK_API::JKFileAdminstrator object;
    object.setSocket(IP, PORT);
    int result = object.delete_file(file_name, version);
    if (0 == result)
    {
        cout << "delete file failed: " << object.show_error() << endl;
        return -1;
    }
    printf("delete file:%s ok, version %d\n", file_name, version);
    return 0;
}
```

A.10.1.9 创建目录

A.10.1.9.1 接口原型：int create_dir(const char* dir_name);

A.10.1.9.2 接口说明：创建目录。

A.10.1.9.3 参数列表见表 A.151：

表 A.151 create_dir 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
dir_name	const char*	In	含相对路径的目录名	是
返回值	int	Out	=1 成功 =0 失败；	—

A.10.1.9.4 处理过程及返回值说明：客户端调用该接口访问文件服务，创建目录，若返回 0，解析返回的报错信息。

A.10.1.9.5 调用示例：

```
int main(int argc, char **argv)
{
    if(3 != argc)
    {
        printf("create_dir【dir_name】【IP】\n");
        return -1;
    }

    char *dir_name = (char *)argv[1];
    char *IP = (char *)argv[2];

    JK_API::JKFileAdminstrator object;
    object.setSocket(IP, 10012);

    int result = object.create_dir(dir_name);
    if (0 == result)
    {
        cout << "create directory failed: " << object.show_error() << endl;
        return -1;
    }
    cout << "create directory " << dir_name << " ok!" << endl;
    return 0;
```

```
}
```

A.10.1.10 查询目录

A.10.1.10.1 接口原型：int list_dir(const char* dir_name, JKFileInfo& fileinfo);

A.10.1.10.2 接口说明：查询目录。

A.10.1.10.3 参数列表见表 A.152：

表 A.152 list_dir 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
dir_name	const char*	In	含相对路径的目录名	是
fileinfo	JKFileInfo	OUT	目录树	是
返回值	int	Out	=1 成功 =0 失败	—
注：目录树结构详见 A.7.1.6				

A.10.1.10.4 处理过程及返回值说明：客户端调用该接口访问文件服务，查询并返回目录信息，若返回 0，解析返回的报错信息。

A.10.1.10.5 调用示例：

```
void print_info(JK_API::JKFileInfo fileinfo)
{
    cout << fileinfo.filename;
    if (fileinfo.is_dir)
        cout << " is dir" ;
    cout << endl;
}
int main(int argc, char **argv)
{
    if(3 != argc)
    {
        printf("list_dir【dir_name】【IP】\n");
        return -1;
    }
    char *dir_name = (char *)argv[1];
    char *IP = (char *)argv[2];

    JK_API::JKFileAdminstrator object;
    object.setSocket(IP, 10012);

    JK_API::JKFileInfo fileinfo;
    int result = object.list_dir(dir_name, fileinfo);
    if (0 == result)
    {
        cout << "list directory fail: " << object.show_error() << endl;
        return -1;
    }
    for (int i = 0; i < fileinfo.fileinfo_list.size(); i++)
    {
        print_info (fileinfo.fileinfo_list[i]);
        cout << "--------------------" << endl;
    }
    cout << endl;
    return fileinfo.fileinfo_list.size();
}
```

A.10.1.11 删除空目录

A.10.1.11.1 接口原型：int delete_dir(const char* dir_name);

A.10.1.11.2 接口说明：删除空目录。

A.10.1.11.3 参数列表见表 A.153：

表 A.153 delete_dir 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
dir_name	const char*	In	含相对路径的目录名	是
返回值	int	Out	=1 成功 =0 失败	—

A.10.1.11.4 处理过程及返回值说明：客户端调用该接口访问文件服务，删除空目录，若返回 0，解析返回的报错信息。

A.10.1.11.5 调用示例：

```
int main(int argc, char **argv)
{
    if(3 != argc)
    {
        printf("delete_dir【dir_name】【IP】\n");
        return -1;
    }
    char *dir_name = (char *)argv[1];
    char *IP = (char *)argv[2];

    JK_API::JKFileAdminstrator object;
    object.setSocket(IP, 10012);
    int result = object.delete_dir(dir_name);
    if (0 == result)
    {
        cout << "delete directory failed: " << object.show_error() << endl;
        return -1;
    }
    cout << "delete directory " << dir_name << " ok!" << endl;
    return 0;

}
```

A.10.1.12 拷贝目录

A.10.1.12.1 接口原型：int copy_dir(const char* src_dir, const char* dest_dir);

A.10.1.12.2 接口说明：拷贝目录。

A.10.1.12.3 参数列表见表 A.154：

表 A.154 copy_dir 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
src_dir	const char*	In	含相对路径的源目录名	是
dest_dir	const char*	In	含相对路径的目的目录名	是
返回值	int	Out	=1 成功 =0 失败	—

A.10.1.12.4 处理过程及返回值说明：客户端调用该接口访问文件服务，拷贝目录，若返回 0，解析返回的报错信息。

A.10.1.12.5 调用示例：

```
int main(int argc, char **argv)
{
```

```
if(4 != argc)
{
    printf("copy_dir【src_dir】【dest_dir】【IP】\n");
    return -1;
}
char *src_dir = (char *)argv[1];
char *dest_dir = (char *)argv[2];
char *IP = (char *)argv[3];

JK_API::JKFileAdminstrator object;
object.setSocket(IP, 10012);
int result = object.copy_dir(src_dir, dest_dir);
if (0 == result)
{
    cout << "copy directory failed: " << object.show_error() << endl;
    return -1;
}
cout << "copy directory " << dir_name << " ok!" << endl;
return 0;
```

A.10.2 日志服务接口

A.10.2.1 接口原型：int LogWrite(int level, JK_LogInfo *log_info, const char *format, …)；

A.10.2.2 接口说明：该接口用于客户端按照日志级别向服务端 syslogd 写入日志信息，日志级别支持 LOG_EMERG、LOG_ALERT、LOG_NOTICE、LOG_INFO 和 LOG_DEBUG。

A.10.2.3 参数列表见表 A.155：

表 A.155 LogWrite 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
level	int	In	日志级别	是
log_info	JK_LogInfo *	In	日志进程信息	是
format	const char *	In	日志内容	是
返回值	int	Out	<0：日志记录失败； >0：记录成功	—

A.10.2.4 JK_LogInfo 结构如下：

```
struct JK_LogInfo
{
    int flag;   /* 0 - process; 1 - library; */
    char ctx_name[JK_MAXNAMELEN];
    char app_name[JK_MAXNAMELEN];
    char pro_name[JK_MAXNAMELEN];
    char lib_name[JK_MAXNAMELEN];
};
```

A.10.2.5 处理过程及返回值说明：日志记录在~HOME/var/log 目录下，无特殊情况不需要判断返回值。

A.10.2.6 调用示例：

```
using namespace JK_API;
JK_LogInfo log_info;
log_info.flag = 0;   /* 进程写日志 */
strcpy(log_info.ctx_name,”realtime”);
strcpy(log_info.app_name,”scada”);
strcpy(log_info.pro_name,”test”);
ret=LogWrite(LOG_INFO,&log_info,”log_writetest…”;)
```

A.10.3 安全认证服务接口

A.10.3.1 消息类业务认证加密接口

A.10.3.1.1 配置初始化

应满足以下要求：

a） 接口原型：int sec_local_init(char* cfg);

b） 接口说明：安全认证初始化，完成认证配置的初始化，该接口适用于消息类业务。

c） 参数列表见表 A.156：

表 A.156 sec_local_init 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
cfg	char*	In	配置文件绝对路径	是
返回值	int	Out	-1：失败 0：成功	—

d） 接口返回值：返回 0 是成功，返回-1 是失败。

e） 调用示例：

```
char* cfg = "/home/scada/conf/secbusauth.conf";
int ret = sec_local_init(cfg); //初始化
if(ret != 0)
   return ret;
```

A.10.3.1.2 接口调用认证

应满足以下要求：

a） 接口原型：int sec_cert_verify(char* cert_file);

b） 接口说明：对消息总线注册者进行认证，该接口适用于消息类业务。

c） 参数列表见表 A.157：

表 A.157 sec_cert_verify 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
cert_file	char*	In	消息总线注册者的证书	是
返回值	int	Out	-1：失败 0：成功	—

d） 接口返回值：返回 0 是成功，-1 是失败。

e） 调用示例：

```
char* cert_file = "/home/scada/conf/sec_certs/AppClient_01.cer";
int ret = sec_cert_verify(cert_file);
if(ret != 0)
   return ret;
```

A.10.3.1.3 安全认证

应满足以下要求：

a） 接口原型：int sec_msgbus_auth(char* password, char* msg_mode);

b） 接口说明：向安全认证服务进行双向认证，该接口适用于消息类业务。

c） 参数列表见表 A.158：

表 A.158 sec_msgbus_auth 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
password	char*	In	P12 文件的保护口令	是
msg_mode	char*	In	消息传输模式，局域消息总线	是

			认证填 01，广域消息总线认证填 03	
返回值	int	Out	-1: 未知错误 0：成功	—

d） 接口返回值：返回 0 是成功，非 0 是失败。

e） 调用示例：

```
char* pwd = "test1234567890";
char* mode = "01";
   int ret = sec_msgbus_auth(pwd, mode);

if(ret != 0)
   return ret;
```

A.10.3.1.4 密钥更新及保活

应满足以下要求：

a） 接口原型：void sec_msgbus_server(int port);

b） 接口说明：服务函数，用于保活和更新安全认证服务的密钥因子，该接口适用于消息类业务。

c） 参数列表见表 A.159：

表 A.159 sec_msgbus_server 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
port	int	In	保活端口号，一般填 6062	是
返回值	void	Out	无	—

d） 接口返回值：无。

e） 调用示例：

```
//另起一个进程或线程，用于保活和更新密钥因子
pthread_t sec_pt;
int ret = pthread_create(&sec_pt, NULL, sec_msgbus_server, NULL);
```

A.10.3.1.5 读取密钥

应满足以下要求：

a） 接口原型：int sec_get_key_mem(char* ip, char* key);

b） 接口说明：根据 IP 读取密钥因子，并生成密钥，该接口适用于消息类业务。

c） 参数列表见表 A.160：

表 A.160 sec_get_key_mem 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
ip	char*	In	消息代理的 IP	是
key	char*	Out	该消息代理的密钥	是
返回值	int	Out	-1：失败 0：成功	—

d） 接口返回值：返回 0 是成功，-1 是失败。

e） 调用示例：

```
char* ip = "192.168.1.1";
char* key[32];
int ret = sec_get_key_mem(ip, key);
if(ret != 0)
   return ret;
```

A.10.3.1.6 数据加密

应满足以下要求：

a） 接口原型：int sec_encode_data(char* key, int flag, char* plain_data, int plain_data_len, char* cipher_data, int* cipher_len);

b） 接口说明：数据加密，该接口适用于消息类业务。

c） 参数列表见表 A.161：

表 A.161 sec_ encode_data 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
key	char *	In	加密密钥，16 字节	是
flag	int	In	0 不加密 1 加密	是
plain_data	char *	In	待加密数据	是
plain_data_len	int	In	待加密数据长度	是
cipher_data	char *	Out	加密后的密文	是
cipher_len	int *	Out	密文长度	是
返回值	int	Out	-1：失败 0：成功	—

d） 接口返回值：返回 0 是成功，-1 是失败。

e） 调用示例：

```
char* ip = "192.168.1.1";
char* key[32];
int ret = sec_get_key_mem(ip, key);
if(ret != 0)
   return ret;
char* data = "XXXXX"; //待加密消息
int data_len = strlen(data); //待加密消息长度
char* cip_data[data_len+16]; //密文
int cip_len;
int flag = 1; //0 不加密 1 加密，此值根据消息头中的加密字段值确定
sec_encode_data(key, flag, data, data_len, cip_data, &cip_len);
```

A.10.3.1.7 数据解密

应满足以下要求：

a） 接口原型：int sec_decode_data(char* key, int flag, char* cipher_data, int cipher_len, char* plain_data, int* plain_data_len);

b） 接口说明：数据解密，该接口适用于消息类业务。

c） 参数列表见表 A.162：

表 A.162 sec_decode_data 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
key	char *	In	加密密钥，16 字节	是
flag	int	In	0 不加密 1 加密	是
cipher_data	char *	In	密文	是
cipher_len	int	In	密文长度	是
plain_data	char *	Out	解密后的明文	是
plain_data_len	int *	Out	明文长度	是
返回值	int	Out	-1：失败 0：成功	—

d） 接口返回值：返回 0 是成功，-1 是失败。

e） 调用示例：

```
char* ip = "192.168.1.1";
char* key[32];
int ret = sec_get_key_mem(ip, key);
if(ret != 0)
   return ret;
char* recv_data = "XXXXX";  //接收到的密文数据
int recv_data_len;
char* data[recv_data_len];  //明文
int data_len = 0;  //明文长度
int flag = 1; //0 不加密 1 加密，此值根据消息头中的加密字段值确定
sec_decode_data(key, flag, recv_data, recv_data_len, data, &data_len);
```

A.10.3.1.8 消息总线调用安全认证接口示例

```
//消息总线启动时添加的代码，伪代码
main(){
   cfg = "/home/scada/conf/secbusauth.conf";
   int ret = sec_local_init(cfg); //初始化
   if(ret != 0)
      return ret;
   //另起一个进程或线程，用于保活和更新密钥因子
   pthread_t sec_pt;
   int ret = pthread_create(&sec_pt, NULL, sec_msgbus_server, NULL);

   //向安全认证服务发起认证，认证后拿到所有已认证消息代理的密钥因子
   ret = sec_msgbus_auth();

   //下面是消息总线自己的代码//
}

//消息总线注册函数内添加的代码，在消息总线注册函数里添加如下代码
messageInit(){

   char* cert_file = "/home/scada/conf/sec_certs/AppClient_01.cer";
   int ret = sec_cert_verify(cert_file);
   if(ret != 0)
      return ret;
  //下面是消息总线自己的代码
}

//加密数据，添加在消息总线发送函数里
char* key[32];
char* ip = "192.168.0.1"; //本机 IP
sec_get_key_mem(ip, key);  //生成密钥
char* data = "XXXXX";  //消息代理从内存中读取到的应用自己的明文数据
int data_len = strlen(data);
char* cip_data[data_len+16];  //密文
int cip_len;
int flag = 1;  //0 不加密 1 加密，此值根据消息头中的加密字段值确定
sec_encode_data(key, flag, data, data_len, cip_data, &cip_len);

//解密数据，添加在消息总线接收函数里
```

```
char* key[32];
char* ip = "192.168.0.1"; //消息发送者的 IP
sec_get_key_mem(ip, key); //生成密钥
char* recv_data = "XXXXX"; //消息代理接收到的密文数据
int recv_data_len;
char* p_data[recv_data_len]; //明文
int p_data_len = 0; //明文长度
int flag = 1; //0 不加密 1 加密，此值根据消息头中的加密字段值确定
sec_decode_data(key, flag, recv_data, recv_data_len, p_data, &p_data_len);
```

A.10.3.2 服务类业务认证加密接口

A.10.3.2.1 安全认证配置初始化

A.10.3.2.1.1 接口原型：int sec_service_local_init(int verify_token=0);

A.10.3.2.1.2 接口说明：安全认证初始化，完成认证配置的初始化，该接口适用于服务类业务。

A.10.3.2.1.3 参数列表见表 A.163：

表 A.163 sec_service_local_init 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
verify_token	int	In	是否认证	是
返回值	int	Out	-1：失败 0：成功	—

A.10.3.2.1.4 接口返回值：返回 0 是成功，返回-1 是失败。

A.10.3.2.1.5 调用示例：

```
int verify_token = 1; // 0 不启用服务认证 1 启用服务认证
int ret = sec_service_local_init(verify_token); //初始化，只需执行一次
if(ret != 0)
  return ret;
```

A.10.3.2.2 安全认证

A.10.3.2.2.1 接口原型：int sec_service_auth(char* srv_name, char* firm_name, char* pwd);

A.10.3.2.2.2 接口说明：向安全认证服务进行双向认证，该接口适用于服务类业务。

A.10.3.2.2.3 参数列表见表 A.164：

表 A.164 sec_service_auth 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
srv_name	char*	In	软件客户端写功能模块名，软件服务端写服务名	是
firm_name	char*	In	厂商名	是
pwd	char*	In	P12 文件的保护口令	是
返回值	int	Out	-1: 未知错误 0：成功 1002：连接服务器失败 1004：认证信息加密失败 1005：认证信息解密失败 1006：认证结果加密失败 1007：认证结果解密失败	—

A.10.3.2.2.4 接口返回值：返回 0 是成功，非 0 是失败。

A.10.3.2.2.5 调用示例：

```
char* srv_name = "xxx"; // 软件客户端模块写模块名，后台服务写服务名
char* firm_name = "kedong"; // 写你自己的厂商名
```

```
char* pwd = "1234"; // 服务 P12 文件的保护口令
ret = sec_service_auth(srv_name, firm_name, pwd); //认证，必须执行
if(ret !=0)
  return 0; //认证失败，自动退出
```

A.10.3.2.3 服务请求加密（带使用者）

A.10.3.2.3.1 接口原型：int sec_service_encode_request_user(int flag, char* req_data, int req_len, char* cip_req, int* cip_req_len, char* serviceInfo);

A.10.3.2.3.2 接口说明：对服务请求报文进行安全加固，将原始请求报文封装成安全报文，该接口适用于服务类业务。

A.10.3.2.3.3 参数列表见表 A.165：

表 A.165 sec_service_encode_request_user 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
flag	int	In	0 不加密报文 1 加密报文	是
req_data	char *	In	服务请求数据	是
req_len	int	In	服务请求数据长度	是
cip_req	char *	Out	安全报文	是
cip_req_len	int *	Out	安全报文长度	是
serviceInfo	char *	In	服务请求者信息	是
返回值	int	Out	-1: 未知错误 0：成功 1206：未找到 Token	—

A.10.3.2.3.4 接口返回值：返回 0 是成功，非 0 是失败。

A.10.3.2.3.5 调用示例：

```
int flag = 1; // 是否加密请求报文，0 表示不加密、1 表示加密
char cip_req[req_len+572]; //安全加固后的新报文,最大长度为原始请求报文长度+572
int cip_req_len = 0;  //新报文长度
char* serviceInfo = "xxx_kedong";  //使用者
//加固请求报文，生成新报文
int ret = sec_service_encode_request_user(flag, req_data, req_len, cip_req,
&cip_req_len,serviceInfo);
```

A.10.3.2.4 服务请求加密

A.10.3.2.4.1 接口原型：int sec_service_encode_request(int flag, char* req_data, int req_len, char* cip_req, int* cip_req_len,);

A.10.3.2.4.2 接口说明：对服务请求报文进行安全加固，将原始请求报文封装成安全报文，该接口适用于服务类业务。

A.10.3.2.4.3 参数列表见表 A.166：

表 A.166 sec_service_encode_request 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
flag	int	In	0 不加密报文 1 加密报文	是
req_data	char *	In	服务请求数据	是
req_len	int	In	服务请求数据长度	是
cip_req	char *	Out	安全报文	是
cip_req_len	int *	Out	安全报文长度	是
返回值	int	Out	-1: 未知错误 0：成功 1206：未找到 Token	—

A.10.3.2.4.4 接口返回值：返回 0 是成功，非 0 是失败。
A.10.3.2.4.5 调用示例：

```
int flag = 1; // 是否加密请求报文，0 表示不加密、1 表示加密
char cip_req[req_len+572]; //安全加固后的新报文,最大长度为原始请求报文长度+572
int cip_req_len = 0;  //新报文长度
//加固请求报文，生成新报文

int ret = sec_service_encode_request(flag, req_data, req_len, cip_req, &cip_req_len);
```

A.10.3.2.5 服务请求解密

A.10.3.2.5.1 接口原型：int sec_service_decode_request(char* cip_req, int cip_req_len, int* flag, char* key, char* req_data, int* req_len);
A.10.3.2.5.2 接口说明：对安全报文进行验证和解密，返回原始的请求报文，该接口适用于服务类业务。
A.10.3.2.5.3 参数列表见表 A.167：

表 A.167 sec_service_decode_request 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
cip_req	char *	In	安全报文	是
cip_req_len	int	In	安全报文长度	是
flag	int *	Out	0 不加密报文 1 加密报文	是
key	char *	Out	加密密钥	是
req_data	char *	Out	原始请求报文	是
req_len	int *	Out	原始请求报文长度	是
返回值	int	Out	-1: 未知错误 0：成功 1202：Token 超时失效 1203：Token 验签失败 1301：时间戳超时 1302：nonce 已存在 1303：安全报文签名验签失败	—

A.10.3.2.5.4 接口返回值：返回 0 是成功，非 0 是失败。
A.10.3.2.5.5 调用示例：

```
//验证 token+解密请求报文
int flag = 0;
char key[32]] = {0};
char* responseBuffer = "xxxx"; //原始服务请求
int responselen = strlen(responseBuffer);
char req_data[requestlen];
int req_len = 0;
int ret = sec_service_decode_request(requestBuffer, requestlen, &flag, key, req_data,
&req_len);
if(ret != 0){  //如果出错，返回错误码
   return ret;
}
```

A.10.3.2.6 读取密钥（带使用者）

A.10.3.2.6.1 接口原型：int get_srv_key_user(char* key,char* serviceInfo);
A.10.3.2.6.2 接口说明：通过该接口获取自己的加密密钥，该接口适用于服务类业务。
A.10.3.2.6.3 参数列表见表 A.168：

表 A.168 get_srv_key_user 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填

key	char *	Out	加密密钥，16 字节	是
serviceInfo	char *	In	服务请求者信息	是
返回值	int	Out	-1：失败 0：成功	—

A.10.3.2.6.4 接口返回值：返回 0 是成功，-1 是失败。

A.10.3.2.6.5 调用示例：

```
char key[32]] = {0};
char* serviceInfo = "xxx_kedong"; //使用者
get_srv_key_user(key,serviceInfo )
```

A.10.3.2.7 读取密钥

A.10.3.2.7.1 接口原型：int get_srv_key(char* key);

A.10.3.2.7.2 接口说明：通过该接口获取自己的加密密钥，该接口适用于服务类业务。

A.10.3.2.7.3 参数列表见表 A.169：

表 A.169 get_srv_key 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
key	char *	Out	加密密钥，16 字节	是
返回值	int	Out	-1：失败 0：成功	—

A.10.3.2.7.4 接口返回值：返回 0 是成功，-1 是失败。

A.10.3.2.7.5 调用示例：

```
char key[32]] = {0};
get_srv_key(key );
```

A.10.3.2.8 数据加密

A.10.3.2.8.1 接口原型：int sec_service_encode_data(char* key, int flag, char* plain_data, int plain_data_len, char* cipher_data, int* cipher_len);

A.10.3.2.8.2 接口说明：数据加密，该接口适用于服务类业务。

A.10.3.2.8.3 参数列表见表 A.170：

表 A.170 sec_service_ encode_data 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
key	char *	In	加密密钥，16 字节	是
flag	int	In	0 不加密 1 加密	是
plain_data	char *	In	待加密数据	是
plain_data_len	int	In	待加密数据长度	是
cipher_data	char *	Out	加密后的密文	是
cipher_len	int *	Out	密文长度	是
返回值	int	Out	-1：失败 0：成功	—

A.10.3.2.8.4 接口返回值：返回 0 是成功，-1 是失败。

A.10.3.2.8.5 调用示例：

```
char* responseBuffer = "xxxx"; //原始服务请求
int responselen = strlen(responseBuffer);
char cip_data[res_len+16];
int cip_len = 0;
ret = sec_service_encode_data(key, flag, responseBuffer, responselen, cip_data, &cip_len);
```

A.10.3.2.9 数据解密

A.10.3.2.9.1　接口原型：int sec_service_decode_data(char* key, int flag, char* cipher_data, int cipher len, char* plain_data, int* plain_data_len);
A.10.3.2.9.2　接口说明：数据解密，该接口适用于服务类业务。
A.10.3.2.9.3　参数列表见表 A.171：

表 A.171 sec_service_decode_data 原语参数列表

接口参数	类型	输入/输出	参数说明	是否必填
key	char *	In	加密密钥，16 字节	是

表 A.128（续）

接口参数	类型	输入/输出	参数说明	是否必填
flag	int	In	0 不加密 1 加密	是
cipher_data	char *	In	密文	是
cipher_len	int	In	密文长度	是
plain_data	char *	Out	解密后的明文	是
plain_data_len	int *	Out	明文长度	是
返回值	int	Out	-1：失败 0：成功	—

A.10.3.2.9.4 接口返回值：返回 0 是成功，-1 是失败。
A.10.3.2.9.5 调用示例：

```
char key[32]] = {0};
get_srv_key(key );
char plain_data[res_len];
int plain_len = 0;
int ret = sec_service_decode_data(key, flag, res_data, res_len, plain_data, &plain_len);
```

A.10.3.2.10　服务交互调用安全认证接口示例

应满足以下要求：
a）服务端：

```
//服务端
#inclide "jk_services.h" //服务总线头文件
#include "jk_secservicebus.h"//安防认证加密头文件

//Do 为回调函数
void *Do(char *requestBuffer,int requestlen,char **responseBuffer,int *responselen)
{
        /* 需要添加的安全代码 start */
        //验证 token+解密请求报文
        int flag = 0;
        char key[32] = {0};
        char req_data[requestlen];
        //收到请求报文，使用安全解密接口示例
        int req_len = 0;
        int ret = sec_service_decode_request(requestBuffer, requestlen, &flag, key, req_data, &req_len);
        if(ret != 0){ //如果出错，返回错误码
            return ret;
        }
        /* 需要添加的安全代码 end */

        /* 根据解密出的客户端请求报文，填充服务端响应信息*/
        int res_len = 48;
        *responselen = res_len + 16;
        *responseBuffer = (char *)malloc(*responselen);
```

```
        // 填写应答内容
        strcpy(*responseBuffer,"request response!");

        /* 需要添加的安全代码 start */
        //加密服务端响应信息
        char cip_data[res_len+16];
        int cip_len = 0;
        ret = sec_service_encode_data(key, flag, responseBuffer, responselen,
        cip_data, &cip_len);
        /* 需要添加的安全代码 end */

        memset(*responseBuff, 0x00, *responselen);
        memcpy(*responseBuffer, cip_data, cip_len);
    *responselen = cip_len;

        return (void *)1;
}

int main()
{
        int srv_ip = "xxx.xxx.xxx.xxx"; // 服务 IP
        int srv_port = xxx; // 端口号
        char* srv_name = "xxx"; // 服务名
        ServiceInfo serviceinfo; // 服务连接信息
        inet_pton(AF_INET, srv_ip, &serviceinfo.addr);
        serviceinfo.port = srv_port;
        strcpy(serviceinfo.servname, srv_name);

        /* 需要添加的安全代码 start */
        int verify_token = 1; // 0 不启用服务认证 1 启用服务认证服务之间调用时，使用安全接口示例
        int ret = sec_service_local_init(verify_token); //初始化，只需执行一次
        if(ret != 0)
            return ret;

        char *srv_name = "demo"; // 服务名
        char* firm_name = "kedong"; // 厂商名
        char* pwd = "1234"; // 服务 P12 文件的保护口令
        ret = sec_service_auth(srv_name, firm_name, pwd); //认证，必须执行
        if(ret !=0)
            return 0; //认证失败，自动退出
        /* 需要添加的安全代码 end */

        ServiceServerInit(serviceinfo, DISPATCH);// 服务初始化
        // 服务注册,参数根据实际情况填
        ServiceRegisterInit(srv_name, context_name, app_name, servid, port, 0);
        ServiceDispatch(serviceinfo, 2, Do); // 服务分发

        while(1)
        {
            sleep(10);
        }
```

```
        return 0;
}
```

b）客户端：

```
//客户端
#inclide "jk_services.h" //服务总线头文件
#include "jk_secservicebus.h"//安防认证加密头文件

int main()
{
        /* 需要添加的安全代码 start */
        int verify_token = 1; // 0 不启用服务认证 1 启用服务认证
        int ret = sec_service_local_init(verify_token); //初始化，只需执行一次
        if(ret != 0)
            return ret;

        char *srv_name = "demo";//模块名
        char* firm_name = "kedong"; // 厂商名
        char* pwd = "1234"; // 服务 P12 文件的保护口令
        ret = sec_service_auth(srv_name, firm_name, pwd); //认证，必须执行
        if(ret !=0)
            return 0; //认证失败，自动退出

        char req_data[20]="xxx"; //请求报文
        int req_len = 20; //请求报文长度
        char *res_data; //响应报文
        int res_len; //响应报度
        int flag = 1; // 是否加密请求报文， 0 表示不加密、 1 表示加密
        char cip_req[req_len+572]; //安全加固后的新报文,最大长度为原始请求报文长度+572
        int cip_req_len = 0; //新报文长度
        //加固请求报文，生成新报文
        int ret = sec_service_encode_request(flag, req_data, req_len, cip_req, &cip_req_len);
        /* 需要添加的安全代码 end */

        ServiceInfo serviceinfo; //初始化服务连接信息，各个字段需要填充
        Handle handle = HANDLE_INIT; //初始化句柄
        ret = serviceRequestSync(serviceinfo, cip_req, cip_req_len, 10, &res_data, &res_len, &handle);
        if(0 > ret) {
            fprintf(stderr, "serviceRequestSync error : %s\n", _service_errdes);
        }

        //接收到服务端返回的 res_data 和 res_len 后，需要解密响应数据
        /* 需要添加的安全代码 start */
        char key[32] = {0};
        get_srv_key(key);
        char plain_data[res_len];
        int plain_len = 0;
        int ret = sec_service_decode_data (key, flag, res_data, res_len, plain_data, &plain_len);
        /* 需要添加的安全代码 end */
```

```
        serviceRequestFree(handle); //释放请求服务函数申请的资源
        serviceHandleFree(handle);// 释放请求句柄

        return 0;
}
```

A.10.4 文语服务接口（JAVA）

A.10.4.1 接口原型：AlarmSound.playToFile(String soundContent, String soundPath,String ip,String port)。

A.10.4.2 接口说明：输入语音播放内容及语音文件路径合成语音文件。见表 172。

表 A.172

接口参数	类型	输入/输出	参数说明	是否必填
soundContent	String	In	语音播放内容	是
soundPath	String	In	待合成语音文件绝对路径	是
ip	String	In	文语服务程序所在节点 ip	是
port	String	In	文语服务程序端口	是
返回值	boolean	Out	语音合成结果，返回 true 表示成功合成语音文件；返回 false 表示未能合成语音文件	—

A.10.4.3 处理过程及返回值说明：调用该接口后，根据输入的播放内容及语音文件路径合成语音文件。

A.10.4.4 调用示例：

```
boolean result = AlarmSound.playToFile("500kV 测试站 5012 开关分闸", "/home/scada/sound/temp.wav", "127.0.0.1", "1234");
 if(result)
 {
   play("/home/scada/sound/temp.wav");
 }
```

A.10.5 文语服务接口（C++）

A.10.5.1 接口原型：int sendAndRecv(const char *sendMsg, const char *&recvMsg, int &length, int timeout);

A.10.5.2 接口说明：发送消息到服务，并等待消息返回。

A.10.5.3 参数列表见表 A.173：

表 A.173 sendAndRecv ()参数列表

接口参数	类型	输入/输出	参数说明	是否必填
const char * sendMsg	Char*	In	发送的文字信息	是
char *& recvMsg	Char*	Out	接收的语音二进制数据	是
length	int	Out	接收的语音二进制数据长度	
int timeout	int	In	超时时间	否
返回值	int	Out	-1：失败 0：成功	—

A.10.5.4 接口返回值：返回 0 是成功，返回-1 是失败。

A.10.5.5 调用示例：

```
int main(int argc, char** argv)
{
    const char *recvMsg;
    int timeout = 300;
    int length = 0;

    CJKMMIClient client;
    string inputtext = "测试文语服务; //文字信息

    int ret = client.sendAndRecv(inputtext.c_str(), recvMsg, length, timeout);
  printf( "messageCallback got recvMsg length %d\n", length);
    //转写音频文件
    free(recvMsg);
    return 0;
}
```

A.10.6 审计服务接口

A.10.6.1 接口原型：int sendAuditLog(const JK_AUDIT_LOG_STRU& audit_log_msg);

A.10.6.2 接口说明：发送审计日志到服务，并等待消息返回。

A.10.6.3 参数列表见表 A.174：

表 A.174 sendAuditLog ()参数列表

接口参数	输入/输出	参数说明	是否必填
JK_AUDIT_LOG_STRU& audit_log_msg	In	审计日志报文	是
返回值	Out	=0：成功 <0：失败	返回值

A.10.6.4 参数说明：

```
typedef vector< JK_ONE_AUDIT_LOG_STRU > JK_AUDIT_LOG_STRU;
struct JK_ONE_AUDIT_LOG_STRU
{
  char username[64];          //用户名
  long datetime;              //审计时间
  char auditsubject[64];      //审计主体
  char auditobject[64];       //审计客体
  int  audittype;             //审计类型：1 操作；2 配置；3 查阅
  int  auditlevel;            //审计级别：1 紧急；2 重要；3 次要；4 一般
  char auditdesc[200];        //审计日志内容
  int result;                 //审计操作结果：1 成功；0 失败
} ;
```

A.10.6.5 调用示例：

```
int main(int argc, char** argv)
{
    CJKAuditLog audit_log_op;  // CJKAuditLog 审计日志接口类对象建议尽量复用，因为每次构造该对象都会初始化一个网络通信客户端，不宜频繁进行构造和析构
    JK_AUDIT_LOG_STRU audit_logs;
     JK_ONE_AUDIT_LOG_STRU one_log;
    one_log.xxx=xxx; //结构体字段赋值
```

```
        audit_logs.push_back(one_log);
    int ret = audit_log_op.sendAuditLog (audit_logs);
     return 0;
}
```

A.11 告警服务

A.11.1 发送告警

A.11.1.1 接口原型：int SendAlarmToServer(jkmessage_invocation* m_pMesInv,
const JK_APP_TO_WARN_SERVICE_MESSAGE_STRU& app_to_warn_service_msg)；

A.11.1.2 接口说明：应用通过该接口向平台发送告警信息，发送的信息体内容可按需构建。

A.11.1.3 参数列表见表 A.175：

表 A.175

接口参数	输入/输出	参数（返回值）说明	备注
jkmessage_invocation* m_pMesInv	In	消息总线对象的指针，相见消息总线附录。	—
JK_APP_TO_WARN_SERVICE_MESSAGE_STRU& app_to_warn_service_msg	In	发送告警的具体内容，可根据应用需求自定义存储数据结构	—
返回值	Out	>=0：成功 <0：失败	—

A.11.1.4 参数说明：

```
typedef vector<UDataValue> JK_SEQ_FIELD_VALUE;
typedef vector<JK_ONE_WARN_MESSAGE_STRU> JK_SEQ_WARN_MESSAGE_TYPE;
struct JK_ONE_WARN_MESSAGE_STRU
{
   int warn_type;  //告警类型
   int app_no;    //应用号
   long node_id;   //节点 ID

   unsigned char is_restrain;  //0 不抑制; 1 抑制
   unsigned char op_type;     //默认为 0
   int  sound_table_key_field_order_no;  //

   int reservered_1;  //保留字段
   int reservered_2;  //保留字段
   JK_SEQ_FIELD_VALUE   seq_field_info;   //登录表
} ;
struct  JK_APP_TO_WARN_SERVICE_MESSAGE_STRU
{
   long warn_num;
   JK_SEQ_WARN_MESSAGE_TYPE seq_warn_message;
};//应用发给告警服务的告警通知消息结构接口返回值：
```

返回值	说明	备注
>=0	成功	—
-1	失败	—

A.11.1.5 应用扩展增加的告警登录表，必须具备以下属性，可在此基础上按需扩展应用所需属性，见表 A.176：

表 A.176

域中文名称	域英文名称	数据类型	长度
时间	occur_time	datetime	8
关键字 ID	key_id	long	8
厂站 ID	st_id	long	8
间隔 ID	bay_id	long	8
设备 ID	dev_id	long	8
告警内容	content	Varchar	200
状态	status	int	4

A.11.1.6 发送告警示例如下：

应用按需定义自己的告警表结构，示例结构如下

数据名称	标识	类型	备注	
时间	occur_time	datetime	必须具备属性	
KEY_ID	key_id	long	必须具备属性	
厂站 ID	st_id	long	必须具备属性	
间隔 ID	bay_id	long	必须具备属性	
设备 ID	dev_id	long	必须具备属性	
内容	content	Varchar(200)	必须具备属性	
状态	status	int	必须具备属性	分配给应用一个告警类型，可以定义多个告警状态分类，平台可根据告警类型+告警状态配置告警方式，告警查询工具也可基于状态做分类查询
告警定制分类	customized_group	int	必须具备该字段	平台统一管理属性，应用发送告警时不需填写
告警确认状态	confirm_status	int	必须具备该字段	平台统一管理属性，应用发送告警时不需填写
确认时间	confirm_time	datetime	必须具备该字段	平台统一管理属性，应用发送告警时不需填写
确认用户	confirm_user_id	int	必须具备该字段	平台统一管理属性，应用发送告警时不需填写
确认节点	confirm_node_id	long	必须具备该字段	平台统一管理属性，应用发送告警时不需填写

应用发送告警示例程序：

JK_APP_TO_WARN_SERVICE_MESSAGE_STRU app_to_warn_service_msg;

app_to_warn_service_msg.warn_num = 1; //一次发送告警的数目

app_to_warn_service_msg.seq_warn_message.length(1); // seq_warn_message 的长度为 1

app_to_warn_service_msg.seq_warn_message[0].warn_type = TEST_WANR_TYPE; //告警类型，平台统一分配，并与应用告警登录表关联

app_to_warn_service_msg.seq_warn_message[0].app_no = AP_SCADA; //所属应用

app_to_warn_service_msg.seq_warn_message[0].is_restrain = 0; //是否告警抑制，应用发送告警时，根据设备状态维护此属性

app_to_warn_service_msg.seq_warn_message[0].seq_field_info.length(6); //应用存储告警的域个数

以下发送顺序及内容与应用自定的存储结构一致。

app_to_warn_service_msg.seq_warn_message[0].seq_field_info[0].c_time(tv_sec); //时间

app_to_warn_service_msg.seq_warn_message[0].seq_field_info[1].c_long();//key_id

app_to_warn_service_msg.seq_warn_message[0].seq_field_info[2].c_long(67890); //st_id

app_to_warn_service_msg.seq_warn_message[0].seq_field_info[3].c_long(12345); //bay_id

```
app_to_warn_service_msg.seq_warn_message[0].seq_field_info[4].c_string("2008-12-18 16:33:00  安宜变 应用测试发送告警  状态 1"); //告警内容
app_to_warn_service_msg.seq_warn_message[0].seq_field_info[5].c_int(1); //应用自定义状态
JKIAlarmClientInterface alarm_op;
int ret = alarm_op.SendAlarmToServer(&m_MsgBus, app_to_warn_service_msg); //发送告警
```

A.11.2 告警订阅

通过消息总线的方式定义接收告警：

a） 消息通道号：JK_API_MSGCHAN_S_WARN_INFORM；

b） 消息事件类型：JK_API_MSGTYPE_S_WARN_INFORM；

c） 消息接口文件 jk_warn_message_new_m.h、jk_warn_message_new_m.cpp，消息文件内容如下：

```
struct jk_warn_msg
{
   short   type;
   unsigned char   op_type;
   MLang::STRING   warn_str;/*告警内容*/
   MLang::STRING   warn_key_str;/*告警唯一关键字*/
   MLang::Long occur_time;
   short   msec_time;
   MLang::Long key_id;
   MLang::Long fac_id;
   MLang::Long bay_id;
   MLang::Long node_id;
   MLang::Long user_id;
   MLang::Long resp_id;
   MLang::Long status;
   int app_no;
   int level;

   jk_warn_msg();
   jk_warn_msg(const jk_warn_msg&);
   jk_warn_msg&operator=(const jk_warn_msg&);
   void __write(MLang::OutputStream&__os)const;
   void __read(MLang::InputStream&__is);
};
typedef MLang::VECTOR<jk_warn_msg> JK_SEQ_WARN_MESSAGE;
struct jk_warn_client_msg
{
   int warn_num;
```

```
    JK_SEQ_WARN_MESSAGE    seq_warn;
    jk_warn_client_msg();
    jk_warn_client_msg(const jk_warn_client_msg&);
    jk_warn_client_msg&operator=(const jk_warn_client_msg&);
    void __write(MLang::OutputStream&__os)const;
    void __read(MLang::InputStream&__is);
};
```

注：其中告警等级（level）对应的显示值和实际值，对应菜单定义表“通用告警等级”菜单；告警记录索引（warn_key_str），是每条告警的唯一标识。

A.11.3 告警类型

从平台告警通道订阅的告警类型包括但不限于以下内容：

```
JK_YX_BW_WARN_TYPE          1    /* 遥信变位 */
JK_YC_OVER_WARN_TYPE        2    /* 遥测越限 */
JK_YX_SOE_WARN_TYPE         3    /* SOE */
JK_OP_YX_WARN_TYPE          5    /* 遥信操作 */
JK_OP_YC_WARN_TYPE          6    /* 遥测操作 */
JK_OP_TOKEN_WARN_TYPE       8    /* 置牌操作 */
JK_OP_CTRL_WARN_TYPE        7    /* 控制操作 */
```

A.11.4 光字告警确认

应用通过消息总线向平台发送光字告警确认消息：

a） 主设备光字牌确认发送通道号：JK_API_MSGCHAN_S_MAIN_LIGHT_CONFIRM；

b） 主设备光字牌确认消息事件号：JK_API_MSGTYPE_S_MAIN_LIGHT_CONFIRM；

c） 辅设备光字牌确认发送通道号：JK_API_MSGCHAN_S_AUX_LIGHT_CONFIRM；

d） 辅设备光字牌确认消息事件号：JK_API_MSGTYPE_S_AUX_LIGHT_CONFIRM。

e） 消息定义文件：jk_warn_confirm_msg_sync_m.h,jk_warn_confirm_msg_sync_m.cpp,消息结构如下：

```
struct jk_tag_warn_confirm_data
{
    MLang::Long l_key_id;              // yuliu
    int   i_warn_type;                 //YX_BW_WARN_TYPE
    int   i_warn_status;       //动作/复归
    MLang::Long l_dev_id;              //光子遥信 id


    void __write(MLang::OutputStream&__os)const;
    void __read(MLang::InputStream&__is);
};
typedef MLang::VECTOR<jk_tag_warn_confirm_data> JK_SEQ_TAG_DATA;
struct jk_tag_warn_confirm_sync
{
```

```
    int   i_op_type; //填 1
    MLang::Long l_node_id;
    MLang::Long l_user_id;
    MLang::Long l_confirm_time;

    JK_SEQ_TAG_DATA    tag_confirm_seq;

    jk_tag_warn_confirm_sync();
    jk_tag_warn_confirm_sync(const jk_tag_warn_confirm_sync&);
    jk_tag_warn_confirm_sync& operator= (const jk_tag_warn_confirm_sync&);
    void __write(MLang::OutputStream&__os)const;
    void __read(MLang::InputStream&__is);
};
```

A.11.5 告警确认

通过消息总线的方式发送和接收告警确认消息：

a） 发送告警确认通道号：JK_API_MSGCHAN_S_WARN_CONFIRM。通过指定节点方式发送至SCADA 主机；

b） 接受确认消息通道号：JK_API_MSGCHAN_R_WARN_ CONFIRM。通过广播方式发送；

c） 消息事件类型：JK_API_MSGTYPE_S_WARN_ CONFIRM；

d） 消息接口文件 jk_warn_message_new_m.h、jk_warn_message_new_m.cpp，消息文件内容如下：

```
struct jk_warn_confirm_info
{
    short   type;
    MLang::STRING   warn_str;/*告警内容*/
    MLang::STRING   warn_key_str;/*告警唯一关键字*/
    jk_warn_confirm_info ();
    jk_warn_confirm_info (const jk_warn_confirm_info &);
    jk_warn_confirm_info &operator=(const jk_warn_confirm_info &);
    void __write(MLang::OutputStream&__os)const;
    void __read(MLang::InputStream&__is);
};
typedef MLang::VECTOR< jk_warn_confirm_info > JK_SEQ_WARN_CONFIRM;
typedef MLang::VECTOR<MLang::Long> KeyIdSeq;
struct jk_warn_confirm_msg
{
    JK_SEQ_WARN_CONFIRM    seq_warn;
    MLang::Long node_id;
```

```
    MLang::Long user_id;
    MLang::Long confirm_time;
    int              yx_confirm_num;
    KeyIdSeq     warn_confirm_keyid;
    jk_warn_confirm_msg();
    jk_warn_confirm_msg(const jk_warn_ confirm_msg&);
    jk_warn_confirm _msg&operator=(const jk_warn_confirm_msg&);
    void __write(MLang::OutputStream&__os)const;
    void __read(MLang::InputStream&__is);
};
```

注：其中告警记录索引（warn_key_str），是每条告警的唯一关键字，同告警订阅结构中的warn_key_str。

A.12 权限服务

A.12.1 基本要求

平台提供统一的权限访问接口，通过调用这些接口访问平台内集成的权限服务，返回执行结果。应用通过接口调用可以判断某个用户是否具备指定功能权限，进行用户的登陆与退出。平台提供统一控制功能，可将各应用界面纳入管理，并集成使用平台统一的登录界面，进行登录与注销操作。

建立权限服务客户端时必须构造一个 CJKPrivAccess 类的对象，由该类的接口负责与权限服务端通信。该类主要提供一系列的接口函数完成权限查询功能。

A.12.2 判断用户是否可用

A.12.2.1 函数原型：short UserIsValid (const char* user_name, int user_id, const char* user_password, const
char* node_name, TKeyID node_id, unsigned char has_password)；

A.12.2.2 功能说明：该接口客户端调用，根据传入的参数判断用户是否是一个可以使用的用户。

A.12.2.3 参数列表见表 A.177：

表 A.177 UserIsValid 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
const char* user_name	char*	In	用户名称	是
int user_id	int	In	用户 ID，user_name、user_id 只要指定其中之一即可	是
const char* user_password	char*	In	用户密码	是
const char* node_name	char*	In	用户所在机器节点名称	是
TKeyID node_id	TKeyID	In	用户所在机器节点 ID，node_name、node_id 只要指定其中之一即可	是
unsigned char has_password	char	In	是否验证用户密码，has_password=1 时需要验证 user_password，否则忽略	是
返回值	short	Out	参考返回值列表	是

A.12.2.4 参数说明：typedef MLang::Long TKeyID

A.12.2.5 头文件：public_m.h

A.12.2.6 调用示例：

```
using namespace JK_API;

//初始化权限客户端 CJKPrivAccess* priv_client = new CJKPrivAccess();
short ret = priv_client->UserIsValid("User",0,"User","net1-1",0,1);
```

A.12.3 查询用户别名

A.12.3.1 函数原型：short getUserAliasByName (const char* user_name, string& user_alias)；

A.12.3.2 功能说明：该接口客户端调用，根据传入的参数用户名称查询用户的别名。

A.12.3.3 参数列表见表 A.178：

表 A.178 getUserAliasByName 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
const char* user_name	char*	In	用户名称	是
string& user_alias	string	Out	用户别名	是
返回值	short	Out	参考返回值列表	是

A.12.3.4 调用示例如下：

```
using namespace JK_API;
CJKPrivAccess* priv_client = new CJKPrivAccess();
short ret = priv_client->getUserAliasByName(user_name.c_str(),user_alias);
```

A.12.4 查询是否具有指定功能号功能

A.12.4.1 函数原型：short HasGivenFunc (const char* user_name, int user_id, const char* user_password, const char* area_name, const char* node_name, TKeyID node_id, int func_id, unsigned char has_password)；

A.12.4.2 功能说明：该接口客户端调用，根据传入的参数判断是否具备指定功能号的功能。

A.12.4.3 参数列表见表 A.179：

表 A.179 HasGivenFunc 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
const char* user_name	char*	In	用户名称	是
int user_id	int	In	用户 ID	是
const char* user_password	char*	In	用户密码	是
const char* area_name	char*	In	责任区名称	是
const char* node_name	char*	In	用户所在机器节点名称	是
TKeyID node_id	TKeyID	In	用户所在机器节点 ID	是
int func_id	int	In	功能 ID，详见 A.9.13	是
unsigned char has_password	unsigned char	In	是否验证用户密码	是
返回值	short	Out	参考返回值列表	是

A.12.4.4 参数说明：typedef MLang::Long TKeyID
A.12.4.5 头文件：public_m.h

A.12.4.6 调用示例：

```
using namespace JK_API;

//初始化权限客户端
CJKPrivAccess* priv_client = new CJKPrivAccess();
//调用 HasGivenFunc 接口
short ret = priv_client->HasGivenFunc("test",0,"","","",45035996273704965,4,1);
```

A.12.5 获得特定用户的详细信息

A.12.5.1 函数原型：short GetGivenUserInfo (const char* user_name, const char* user_password, TKeyID node_id, TUserDefaultInfo2& user_info, unsigned char has_password);
A.12.5.2 功能说明：该接口客户端调用，根据传入的参数返回特定用户的详细信息。
A.12.5.3 参数列表见表 A.180：

表 A.180 GetGivenUserInfo 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
const char* user_name	char*	In	用户名称	是
const char* user_password	char*	In	用户密码	是
TKeyID node_id	TKeyID	In	用户所在机器节点 ID	是
TUserDefaultInfo2& user_info	TUserDefaultInfo2	Out	用户详细信息	是
unsigned char has_password	unsigned char	In	是否验证用户密码	是
返回值	short	Out	参考返回值列表	是

A.12.5.4 参数说明：

```
typedef MLang::Long TKeyID //头文件：public_m.h
// 头文件 priv_server_m.h
struct TUserDefaultInfo2
{
    int   user_id;
    MLang::STRING   user_name; // 用户名称
    MLang::STRING   user_alias; //用户别名
    MLang::STRING   user_desp; // 用户描述
    MLang::STRING   user_pass; //用户密码
    MLang::Long create_date;    // 创建日期
    MLang::Long expire_date;    // 失效日期
    Int remind_days;            // 提醒天数
    Int user_group;             // 用户组
    Int is_leader;              // 是否组长
    unsigned int user_role;  // 用户角色
    char role_priv[201];        // 角色包含指定的功能
};
```

A.12.5.5 调用示例：

```
using namespace JK_API;

//初始化权限客户端
CJKPrivAccess* priv_client = new CJKPrivAccess();
//调用 GetGivenUserInfo 接口
TUserDefaultInfo2 user_info;
short ret=priv_client->GetGivenUserInfo("ddd","",45035996273704965,user_info,1);
```

A.12.6 获得表域的特殊属性

A.12.6.1 函数原型： short HasTabColSpeAttr (const char* user_name, int user_id, const char* user_password,const char* area_name, const char* node_name, TKeyID node_id, const char* table_name, int table_id, const char* column_name, int column_id, int func_id, unsigned char has_password, short& has_func_flag, short& col_spe_flag)；

A.12.6.2 功能说明：该接口客户端调用，根据传入的参数获得用户的默认权限针对给定数据表域的特殊属性。

A.12.6.3 参数列表见表 A.181：

表 A.181 HasTabColSpeAttr 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
const char* user_name	char*	In	用户名称	是
int user_id	int	In	用户 ID	是
const char* user_password	char*	In	用户密码	是
const char* area_name	char*	In	责任区名称	是
const char* node_name	char*	In	用户所在机器节点名称	是
TKeyID node_id	TKeyID	In	用户所在机器节点 ID	是
const char* table_name	char*	In	表中文名	是
int table_id	int	In	表号	是
const char* column_name	char*	In	域中文名	是
int column_id	int	In	域号	是
int func_id	int	In	功能 ID	是
unsigned char has_password	char	In	是否验证用户密码	是
short& has_func_flag	short	Out	用户是否具有指定的功能，返回值为 P_PERMIT 或 P_FORBID	是
short& col_spe_flag	short	Out	用户对所指定表域的特殊属性，返回值为 P_COL_FORBID、P_COL_QUERYONLY、P_COL_UPDATE、P_COL_ADDDEL、	是

			P_COL_EDITABLE 和 P_NO_SPEATTR 中的一个	
返回值	short	Out	参考返回值列表	是

A.12.6.4 调用示例：

```
using namespace JK_API;
//初始化权限客户端
CJKPrivAccess* priv_client = new CJKPrivAccess();
//调用 HasTabColSpeAttr 接口
short has_func_flag,col_spe_flag;
short ret = priv_client->HasTabColSpeAttr("test",0,"","","",45035996273704965,
    "表信息表",0,"应用类型",0,4,1,has_func_flag,col_spe_flag);
```

A.12.7 获得图形的特殊属性

A.12.7.1 函数原型： short HasGrhSpeAttr (const char* user_name, int user_id, const char* user_password,const char* area_name , TKeyID user_node, const char* graph_name, unsigned char has_password,short& grh_spe_flag)；

A.12.7.2 功能说明：该接口客户端调用，根据传入的参数得到用户的默认权限针对给定图形(整体)的特殊属性。

A.12.7.3 参数列表见表 A.182：

表 A.182 HasGrhSpeAttr 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
const char* user_name	char*	In	用户名称	是
int user_id	int	In	用户 ID	是
const char* user_password	char*	In	用户密码	是
const char* area_name	char*	In	责任区名称	是
TKeyID user_node	TKeyID	In	用户所在机器节点 ID	是
const char* graph_name	char*	In	图形名称	是
unsigned char has_password	unsigned char	In	是否验证用户密码	是
short& grh_spe_flag	short	In	用户对所指定图形的特殊属性，返回值为 P_NO_SPEATTR、P_GRH_FORBID、P_GRH_READONLY 和 P_GRH_EDITABLE 中的一个	是
返回值	short	Out	参考返回值列表	是

A.12.7.4 参数说明：typedef MLang::Long TKeyID

A.12.7.5 头文件：public_m.h

A.12.7.6 调用示例：

```
using namespace JK_API;

//初始化权限客户端
CJKPrivAccess* priv_client = new CJKPrivAccess();
short grh_spe_flag;
short ret= priv_client->HasGrhSpeAttr("d5000",0,"","",45035996273704965,
                                      "test2",1,grh_spe_flag);
```

A.12.8 判断是否具有全表/表域的特殊属性

A.12.8.1 函数原型： short isUserHasTabColSpe (const char* user_name)；

A.12.8.2 功能说明：该接口客户端调用，根据传入的参数用户名称判断是否定义了全表/表域的特殊属性。

A.12.8.3 参数列表见表 A.183：

表 A.183 isUserHasTabColSpe 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
const char* user_name	char*	In	用户名称	是
返回值	short	Out	参考返回值列表	是

A.12.8.4 调用示例：

```
using namespace JK_API;

//初始化权限客户端
CJKPrivAccess* priv_client = new CJKPrivAccess();
//调用 isUserHasTabColSpe 接口
short ret = priv_client->isUserHasTabColSpe("test");
```

A.12.9 判断是否具有图形(整体)的特殊属性

A.12.9.1 函数原型：short isUserHasGrhSpe (const char* user_name)；

A.12.9.2 功能说明：该接口客户端调用，根据传入的参数用户名判断是否定义了图形(整体)的特殊属性。

A.12.9.3 参数列表见表 A.184：

表 A.184 isUserHasGrhSpe 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
const char* user_name	char*	In	用户名称	是
返回值	short	Out	参考返回值列表	是

A.12.9.4 调用示例：

```
using namespace JK_API;
//初始化权限客户端
CJKPrivAccess* priv_client = new CJKPrivAccess();
//调用 isUserHasGrhSpe 接口
short ret = priv_client->isUserHasGrhSpe("test");
```

A.12.10 判断是否具有指定功能号功能

A.12.10.1　函数原型：short isRoleHasGivenFunc (const char* role_name, int func_id);
A.12.10.2　功能说明：该接口客户端调用，根据传入的参数判断角色是否具有指定功能号功能。
A.12.10.3　参数列表见表 A.185：

表 A.185 isRoleHasGivenFunc 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
const char* role_name	char*	In	角色名称	是
int func_id	int	In	功能 ID，详见 A.9.13	是
返回值	short	Out	参考返回值列表	是

A.12.10.4 调用示例：

```
using namespace JK_API;

//初始化权限客户端
CJKPrivAccess* priv_client = new CJKPrivAccess();
//调用 isRoleHasGivenFunc 接口
short ret = priv_client->isRoleHasGivenFunc("manager",5);
```

A.12.11　获取全表/表域特殊属性信息

A.12.11.1 函数原型： short HasTabColSpeInfo(const char* user_name, int user_id, const char* user_password,const char* node_name, TKeyID node_id, const char* table_name_eng, int table_id, unsigned char has_password, short &alltab_spe_type, vector<TColSpeInfo2> &col_spes)；
A.12.11.2 功能说明：该接口客户端调用，根据传入的参数得到用户对给定表的全表/表域特殊属性信息。
A.12.11.3 参数列表见表 A.186：

表 A.186 HasTabColSpeInfo 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
const char* user_name	char*	In	用户名称	是
int user_id	int	In	用户 ID	是
const char* user_password	char*	In	用户密码	是
const char* node_name	char*	In	用户所在机器节点名称	是
TKeyID node_id	TKeyID	In	用户所在机器节点 ID	是
const char* table_name_eng	char*	In	表英文名	是
int table_id	int	In	表号	是
unsigned char has_password	unsigned char	In	是否验证用户密码	是
short &alltab_spe_type	short	Out	全表特殊属性信息	是
vector<TColSpeInfo2> &col_spes	vector<TColSpeInfo2>	Out	表域特殊属性信息	是
返回值	short	Out	参考返回值列表	是

A.12.11.4　参数说明：typedef MLang::Long TKeyID

A.12.11.5 头文件：public_m.h

A.12.11.6 调用示例：

```
using namespace JK_API;

//初始化权限客户端
CJKPrivAccess* priv_client = new CJKPrivAccess();
//调用 HasTabColSpeInfo 接口
short alltab_spe_type;
vector<TColSpeInfo2> col_spes;
short ret = priv_client->HasTabColSpeInfo("nari",0,"","scd2-1",0,"",161,0,
alltab_spe_type,col_spes);
```

A.12.12 获取指定功能的权限

A.12.12.1 函数原型：int HasGivenFuncsWithFlag(const char* user_name, int user_id, const char* user_password, const char* node_name, TKeyID user_node, vector<int> func_list, int spe_type, unsigned char has_password, vector<TFuncPriv> &func_result, int &spe_flag, string& user_alias)；

A.12.12.2 功能说明：该接口客户端调用，根据传入的参数得到用户对给定功能的权限。

A.12.12.3 参数列表见表 A.187：

表 A.187 HasGivenFuncsWithFlag 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
const char* user_name	char*	In	用户名称	是
int user_id	int	In	用户 ID	是
const char* user_password	char*	In	用户密码	是
const char* node_name	char*	In	用户所在机器节点名称	是
TKeyID user_node	TKeyID	In	用户所在机器节点 ID	是
vector<int> func_list	vector<int>	In	功能列表	是
int spe_type	int	In	特殊权限类型，P_TABCOL：表域特殊权限；P_GRAPH：图形特殊权限	是
unsigned char has_password	unsigned char	In	是否验证用户密码	是
vector<TFuncPriv> &func_result	vector<TFuncPriv>	Out	功能权限信息	是
int &spe_flag	int	Out	参考返回值列表	是
string& user_alias	string	Out	用户别名	是
返回值	int	Out	参考返回值列表	是

A.12.12.4 参数说明：typedef MLang::Long TKeyID

A.12.12.5 头文件：public_m.h

A.12.12.6 调用示例：

```
using namespace JK_API;
//初始化权限客户端
CJKPrivAccess* priv_client = new CJKPrivAccess();
```

```
//调用 HasGivenFuncsWithFlag 接口
vector<int> func_list;
vector<TFuncPriv> func_result;
int spe_flag;
string      user_alias;
func_list.push_back(3);
func_list.push_back(4);
func_list.push_back(5);
int ret = priv_client->HasGivenFuncsWithFlag("test",0,"","scd2-1",0,func_list,0,0,
func_result,spe_flag,user_alias);
```

A.12.13 获得用户对应的责任区列表

A.12.13.1 函数原型：int GetUserResps(string user_name, map<string,long> &resps);

A.12.13.2 功能说明：调用该接口可获得用户对应的责任区信息，包含责任区名称与责任区 id 对应关系，应用界面可在调用 A.12.14 登录后，调用该接口获取登录用户可切换责任区信息并进行列表展示、选择。

A.12.13.3 参数列表见表 A.188：

表 A.188 GetUserResps 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
user_name	string	In	用户名称	是
resps	map<string,long>	Out	用户可切换责任区名与责任区 id 的对应关系	是
返回值	int	Out	参考返回值列表	是

A.12.13.4 调用示例：

```
using namespace JK_API;
//初始化权限客户端
CJKPrivAccess* priv_client = new CJKPrivAccess();
//调用 GetUserResps 接口
map<string,long> resps;
int ret = priv_client->GetUserResps("nari",resps);
```

A.12.14 QT登录界面框调用

A.12.13.1 函数原型：int GetLoginDialog(string& user_name, int &user_id, int& expire_time);

A.12.13.2 功能说明：调用该接口弹出登录框，登录成功后返回登录用户名、用户 id 和可保持登录状态最大时长（秒），通过该接口登录后不影响工作站总控台已登录用户信息，应用界面工具根据可保持登录状态最大时长与当前时间计算登录失效时间。

A.12.13.3 参数列表见表 A.189：

表 A.189 GetLoginDialog 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
user_name	string	Out	登录用户名	是
user_id	int	Out	登录用户 id	是
expire_time	int	Out	可保持登录状态最大时长（秒）	是
返回值	int	Out	>0 成功；<0 失败	是

A.12.13.4 调用示例：

```
CJKCommonLoginDialog dialog;
```

```
string user_name;
int user_id;
int expire_time;
int ret=dialog. GetLoginDialog (user_name, user_id, expire_time);
```

A.12.15 获取当前节点登录用户

A.12.15.1 函数原型：int GetCurUser (string &user_name, int &user_id);
A.12.15.2 功能说明：调用该接口获取当前工作站总控台已登录用户信息，应用界面调用 A.12.14 登录的用户信息在应用程序端记录，不通过本接口返回。
A.12.15.3 参数列表见表 A.190：

表 A.190 GetCurUser 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
user_name	string	Out	登录用户名	是
user_id	int	Out	登录用户 id	是
返回值	int	Out	>0 成功； =0 未登录； <0 失败；	是

A.12.15.4 调用示例：

```
using namespace JK_API;
//初始化权限客户端
CJKPrivAccess* priv_client = new CJKPrivAccess();
//调用 GetCurUser 接口
string user_name="";
int user_id;
int ret = priv_client->GetCurUser (user_name, user_id);
if(ret<0)
{
    printf("调用接口失败！");
}
else if(ret==0)
{
    printf("总控台未登录！");
            }
else
{
    printf("当前节点总控台登录用户:%s！",user_name.c_str());
}
```

A.12.16 获取当前节点登录责任区

A.12.16.1 函数原型：int GetCurRespValue (long &resp_value, string &resp_name, boolean &is_all_resp);
A.12.15.2 功能说明：调用该接口获取当前工作站总控台登录责任区值与责任区名称，如果 is_all_resp 为 true 则表示全责任区。
A.12.15.3 参数列表见表 A.191：

表 A.191 GetCurRespValue 参数列表

接口参数	类型	输入/输出	参数说明	是否必填
resp_value	long	Out	登录责任区值	是
resp_name	string	Out	责任区名称	是
is_all_resp	boolean	Out	是否为全责任区	是

返回值	int	Out	>0 成功； =0 未登录； <0 失败	是

A.12.15.4 调用示例：

```
using namespace JK_API;
//初始化权限客户端
CJKPrivAccess* priv_client = new CJKPrivAccess();
//调用 GetCurRespValue 接口
long resp_value=-1;
string resp_name;
boolean is_all_resp = false;
int ret = priv_client->GetCurRespValue (resp_value, resp_name, is_all_resp);
if(ret<0)
{
    printf("调用接口失败！");
}
else if(ret==0)
{
    printf("当前节点总控台未登录！");
}
else
{
    if(is_all_resp)
    {
        printf("当前节点总控台登录责任区为：全责任区！");
    }
    else
    {
        printf("当前节点总控台登录责任区值:%ld，责任区名称：%s！",resp_value，
     resp_name.c_str());
     }
}
```

A.12.17 主要权限功能ID

主要权限功能 ID 见表 A.192：

表 A.192 主要权限功能 ID

权限功能定义	功能 ID 号	描述
REAL_DB_READ	4	/* 模型定义读 */
REAL_DB_WRITE	5	/* 模型定义写 */
MODEL_ADMIN	7	/* 模型维护 */
GRAPH_READ	12	/* 画面文件读 */
GRAPH_EDIT	13	/* 画面文件写 */
PRIV_YX_AFFIRM	20	/* 遥信对位 */
PRIV_YK_OP	21	/* 遥控 */
PRIV_YT_OP	22	/* 遥调 */
EXP_FULL_DATABASE	47	/* 商用库备份 */
IMP_FULL_DATABASE	48	/* 商用库恢复 */
PRIV_CTRL_LOCK	81	/* 控制闭锁 */

PRIV_CTRL_GUARDER	82	/* 控制监护 */
WARN_CONFIRM_OP	87	/* 告警窗确认 */

A.12.18 返回值列表

返回值列表见表 A.193：

表 A.193 返回值列表

0	P_NORMAL	正常返回(无特殊意义)
1	P_FORBID	用户不具有指定功能
2	P_PERMIT	用户具有指定功能
3	P_NO_USER	输入的用户不存在
11	P_INVALID	错误的用户名或密码
12	P_EXPIRE	用户已经失效
13	P_NO_NODE	当前用户不拥有指定的节点
14	P_HAS_SPE	当前用户定义了特殊属性
15	P_NO_ROLE	输入的角色不存在
16	P_ROLE_FORBID	角色不具有指定功能
17	P_ROLE_PERMIT	角色具有指定功能
100	P_NO_SPEATTR	未定义特殊属性

A.13 模型修改服务

A.13.1 模型修改接口

基础平台提供统一的模型修改接口，传入修改类型 SQL 调用接口增、删、改模型数据，并返回执行结果。应满足以下要求：

A.13.1.1 接口原型：short ModifyTableBySqls(const TJKSqlModifyRequest& sql_modify_request, SEQJKSqlModifyAnswer& sql_modify_answer);

A.13.1.2 接口说明:根据 sql 语句对商用库进行操作，同时将数据同步到实时态下实时库，主键 id 根据表的主键是否自动生成来处理。

A.13.1.3 参数见表 A.194：

表 A.194 ModifyTableBySqls 参数列表

接口参数	输入/输出	参数（返回值）说明	备注
sql_modify_request	In	填写请求 sql 等信息	sql 语句的关键字段必须大写，比如 insert…..values…要写成 INSERT…VALUS… sql 语句中的表名、域名要与表信息表、域信息表中对应记录保持的一致（大小写上） sql 语句中 INSERT、DELETE 后直

			接跟表名不要跟 INTO、FROM，将 INTO、FROM 舍去 sql 语句结尾处不需要“；”
sql_modify_answer	Out	应答反馈结构体	
返回值	Out	<0 请求失败 =0 请求成功	返回值只是标识了请求成功与否，若想了解每条 sql 语句的操作结果，检测 sql_modify_answer[i].success_flag，等于 0 则成功，不等于 0 则该条 sql 操作失败

参数说明：

```
struct TJKSqlModifyRequest
{
    int app_no; //应用号
    vector<string> seq_sql_request; //上表中要求格式的 sql 语句
};

struct TJKSqlModifyAnswer
{
    int table_no; //表号
    short op_type;//操作类型类型，0 表示 insert ， 1 表示 delete，2 表示 update
    short success_flag;//该条是否执行成功，0 表示成功
    short auto_gen_type;//该表是否为关键字自动生成
    short key_col_id;//关键字域号
    long key_value;//关键字值
    string syntax_err_msg;//错误信息
};
typedef vector<TJKSqlModifyAnswer> SEQJKSqlModifyAnswer;
```

A.13.1.4 调用示例如下：

```
//初始化数据更新客户端
CJKModelModifyClient  modify_modify_client;

//参数声明
TJKSqlModifyRequest         sql_modify_request;
SEQJKSqlModifyAnswer              sql_modify_answer;

//入参赋值
string sql_str1,sql_str2;
sql_str1 = "INSERT test(code,name) VALUES('sql','sql')";
sql_str2 = "INSERT test(code,name) VALUES('sql2','sql2')";
```

```
    sql_modify_request.seq_sql_request.push_back(sql_str1);
    sql_modify_request.seq_sql_request.push_back(sql_str2);

    sql_modify_request.app_no = AF_SCADA;

    //调用 ModifyTableBySqls 接口
    ret = model_modify_client.ModifyTableBySqls(sql_modify_request, sql_modify_answer);
  if(ret=0)//请求成功，检测每条 sql 语句执行的情况
  {
    for(i=0;i<sqlnum;i++)
    {
      if(sql_modify_answer[i]. success_flag !=0)
      {
        printf(" 第 %d 条 sql 出 错 , 错 误 信 息 为 : %s\n",
i ,sql_modify_answer[i].syntax_err_msg.c_str());
      }
    }
  }
  else//请求失败时，返回值小于 0
  {
    printf("请求失败\n");
    return -1;
  }
  printf("请求成功\n");
  return 1;
```

A.13.2 模型变化订阅

新增、修改、删除模型后通过消息总线发送模型修改消息，消息包含操作类型和变化模型 ID，关注的功能可订阅获取模型修改消息。模型变化订阅表、域范围详见 A.2.3 数据字典。

A.13.2.1 接口原型：

```
int MDFInit(std::string& err_msg);
int GetMDFRecordInfo(std::vector<TJKMDFRecordInfo> &mdf_records_vec);
```

A.13.2.2 接口说明：CJKRtdbMdfClient 调用 MDFInit 接口进行模型变化订阅初始化，CJKRtdbMdfClient 实例初始化成功后，循环调用 GetMDFRecordInfo 接口获取与上一次调用之间的模型变化信息。

A.13.2.3 参数见表 A.195 和 A.196：

表 A.195 MDFInit 参数列表

接口参数	输入/输出	参数（返回值）说明	备注
err_msg	Out	接口报错信息	—
返回值	Out	=0 成功，<0 失败	—

表 A.196 GetMDFRecordInfo 参数列表

接口参数	输入/输出	参数（返回值）说明	备注
mdf_records_vec	Out	模型修改消息	—

返回值	Out	>=0 成功，<0 失败	—

参数说明：

```
struct TJKMDFFieldInfo
  {
     short  field_no;//域号
     std::string  field_name; //字段名称
     short  field_type; //字段数据类型
     std::string field_data; //字段值
  };

  struct TJKMDFRecordInfo
  {
     int app_no;//应用号
     int table_no;//表号
    short op_type; //记录操作类型，引用上述 JK_MDF_OP_INSERT、JK_MDF_OP_DELETE（此时column_infos 为空）、JK_MDF_OP_UPDATE
     int key_num; //表关键字个数
     std::vector<TJKMDFFieldInfo> key_infos; //关键字的类型及数值信息
      std::vector<TJKMDFFieldInfo> column_infos; //配置中发生变化的非关键字字段的类型及数值信息
  };
```

相关宏定义：

```
//域数据类型类型
#define JK_MDF_FIELD_TYPE_STRING  1
#define JK_MDF_FIELD_TYPE_UCHAR  2
#define JK_MDF_FIELD_TYPE_SHORT  3
#define JK_MDF_FIELD_TYPE_INT  4
#define JK_MDF_FIELD_TYPE_DATETIME  5
#define JK_MDF_FIELD_TYPE_FLOAT  6
#define JK_MDF_FIELD_TYPE_DOUBLE  7
#define JK_MDF_FIELD_TYPE_KEYID  8
#define JK_MDF_FIELD_TYPE_BINARY  9
#define JK_MDF_FIELD_TYPE_TEXT  10
#define JK_MDF_FIELD_TYPE_IMAGE  11
#define JK_MDF_FIELD_TYPE_APPKEY  12
#define JK_MDF_FIELD_TYPE_APPID  13
#define JK_MDF_FIELD_TYPE_UINT  14
#define JK_MDF_FIELD_TYPE_LONG  15
//记录变化类型
const int JK_MDF_OP_INSERT = 0;
const int JK_MDF_OP_DELETE = 1;
const int JK_MDF_OP_UPDATE = 2;
```

d) 模型变化订阅示例如下：

```
//初始化客户端
CJKRtdbMdfClient   rtdb_mdf_client; //不要每次构造实例，构造一次后循环调用GetMDFRecordInfo 接口获取模型变化信息
string err_msg = "";
int ret = rtdb_mdf_client.MDFInit(err_msg);
if(ret<0)
```

```
{
 cout<<" rtdb_mdf_client init error:"<<err_msg<<<<endl;
 return -1;
}
while(1)
{
   vector<TJKMDFRecordInfo> mdf_res;
   int mdf_info_nums = rtdb_mdf_client.GetMDFRecordInfo(mdf_res);
   for(int mdf_info_index =0 ;mdf_info_index<mdf_info_nums;mdf_info_index++)
   {
      cout<<"no."<<mdf_info_index<<"msg info:"<<endl;
      cout<<"app_no:"<<mdf_res[mdf_info_index].app_no<<endl;
      //模型变化记录逐条、逐个字段判断使用
   }
   sleep(5);
}
```

A.14 操作控制服务接口

操作控制类服务功能，采用服务总线方式实现，各功能用服务号进行区分，传入参数和传出参数，均采用 Json 格式，编码 GBK，顺控操作调用接口采用消息总线方式实现。

a) 各服务所包含的功能点见表 A.197：

表 A.197 控制服务名称

服务名	规范项	备注	分区
aux_op_srv	A.11.1 遥控、设点	辅设备	二区
sca_yk_serv	A.11.1 遥控、设点、调档	主设备、二次设备（压板）	一区
ied_op_service	A.11.4 二次设备服务	二次设备	二区
ied_op104_service	A.11.3 参数定值修改服务接口	二次设备 104 协议	一区
common_op_srv	A.11.5.1 下发一般文件	—	二区
	A.11.5.2 召唤一般文件	—	
	A.11.5.3 调阅文件列表	—	
	A.11.5.4 RPC 模型调阅	—	
	A.11.5.5 RPC 数据调阅	—	
	A.11.5.6 读数据集	—	
—	A.11.5.7 删除数据集	—	—

b) 调用示例：

```
#inclide "jk_services.h" //服务总线头文件
#include "jk_secservicebus.h"//安防认证加密头文件

int main()
{
      /*第一步：开启安全认证*/
      int verify_token = 1; // 0 不启用服务认证 1 启用服务认证
      int ret = sec_service_local_init(verify_token); //初始化，只需执行一次
      if(ret != 0)
   {
      return ret;
   }
```

```
  char *srv_name = "aux_op_srv";//服务名
    char* firm_name = "kedong"; // 厂商名
    char* pwd = "1234"; // 服务 P12 文件的保护口令
    ret = sec_service_auth(srv_name, firm_name, pwd); //认证
    if(ret !=0)
  {
     return -1; //认证失败，自动退出
  }

  /*第二步：定位本地服务，判断服务是否在线*/
    ServiceInfo srvInfo;//初始化服务连接信息
    memset(&srvInfo, 0, sizeof(srvInfo));
    ret = ServiceLocate("realtime", "tmr_omda", "aux_op_srv", &srvInfo, 0);//获取服务
    if(ret < 0) //获取服务失败
    {
        printf("ServiceLocate failed![%d]\n", ret);
        return -1;
    }
    srvInfo.addr = htonl(srvInfo.addr);
  /*第三步：对传入数据进行安全加密*/
    char req_data[1024]="{json}"; //请求报文
    int req_len = strlen(req_data); //请求报文长度
    char *res_data = NULL; //响应报文
    int res_len = 0; //响应报文长度
    int flag = 1; // 是否加密请求报文， 0 表示不加密、 1 表示加密
    char cip_req[req_len+572]; //安全加固后的新报文,最大长度为原始请求报文长度+572
    int cip_req_len = 0; //新报文长度
    ret = sec_service_encode_request(flag, req_data, req_len, cip_req, &cip_req_len);//生成加密
报文
  if(ret<0)//加密失败
  {
    printf("encode data failed!\n");
    return -1;
  }

  /*第四步：同步请求服务*/
    Handle handle = HANDLE_INIT; //初始化句柄
    ret = serviceRequestSync(serviceinfo, cip_req, cip_req_len, 10, &res_data, &res_len,
&handle);
    if(0 > ret) {
        fprintf(stderr, "serviceRequestSync error : %s\n", _service_errdes);
        //如果返回 token 验证失败，需重新认证
    }

    /*第五步：对传出数据进行解密*/
    char key[32] = {0};
    get_srv_key(key);
    char plain_data[res_len];
    int plain_len = 0;
    ret = sec_service_decode_data (key, flag, res_data, res_len, plain_data, &plain_len);
  if(ret<0)//解密失败
  {
```

```
    printf("decode data failed!\n");
  }
  else{...}//解密成功，处理返回数据

  /*第六步：释放服务资源*/
    if(handle!=-1)
  {
        serviceRequestFree(handle); //释放请求服务函数申请的资源
        serviceHandleFree(handle);// 释放请求句柄
  }
    return 0;
}
```

A.14.1 遥控、设点、调档和控制校核

A.14.1.1 接口原型：参考 A.11 的服务调用示例代码。

A.14.1.2 入参键值说明见表 A.198：

表 A.198 遥控、设点、调档入参说明

参数	说明	数据类型	备注
op_type	服务号	String	详见服务号定义
app_no	应用号	String	
key_id	控制目标 ID	String	顺控校核时为间隔 ID
st_id	厂站 ID	String	
ctrl_time	控制当前时间	String	unix 时间戳（秒）
ctrl_type	控制类型	String	1 常规遥控 2 同期遥控 3 无压遥控 4 档位升 5 档位降 6 档位急停 7 设点 8 控制校核 注：顺控校核时，ctrl_type 必须设置为 8
ctrl_event	控制事件	String	1 预置 2 执行 3 撤销 4 直控 101 控制校核(操作互斥、挂牌闭锁) 注：控制校核时，ctrl_event 必须设置为 101
ctrl_value	控制目标值	String	—
ctrl_machine	控制节点	String	—
guarder_machine	监护节点	String	—
ctrl_user	控制用户	String	—
guarder_user	监护用户	String	—
ctrl_info	备注	String	—

A.14.1.3 入参示例：

```
{
    "query_para": {
        "op_type": "10100300",
        "app_no": "1060000",
        "key_id": "123456789",
        "st_id": "147258369",
        "ctrl_time": "1653365532",
        "ctrl_event":"2",
```

```
            "ctrl_type": "0",
            "ctrl_value": "1",
            "ctrl_machine": "jk-scd1",
            "guarder_machine": "jk-scd1",
            "ctrl_user": "test001",
            "guarder_user": "monitor001",
            "ctrl_info": "default"
        }
    }
```

A.14.1.4 出参键值说明见表 A.199：

表 A.199 遥控、遥调、设点出参说明

参数	说明	数据类型	备注
ret_no	控制结果	String	详见错误代码
ret_msg	返回信息	String	—
app_no	应用号	String	—
key_id	控制目标 ID	String	—
ctrl_time	控制当前时间	String	unix 时间戳

A.14.1.5 出参示例：

```
{
    "query_result": {
        "ret_no": "1",
        "ret_msg": "control time out",
        "app_no": "1060000",
        "key_id": "123456789",
        "ctrl_time": "1653365532"
    }
}
```

A.14.2 顺序预置

A.14.2.1 顺序预置入参键值说明

参数	说明	数据类型	备注
op_type	服务号	string	详见服务号定义
app_no	应用号	string	—
key_id	控制目标 ID	string	—
st_id	厂站 ID	string	—
ctrl_time	控制当前时间	string	unix 时间戳
ctrl_event	控制事件	string	1 预置
ctrl_type	控制类型	string	1 常规
ctrl_value	控制目标值	string	—
ctrl_machine	控制节点	string	—
guarder_machine	监护节点	string	—
ctrl_user	控制用户	string	—
guarder_user	监护用户	string	—
chn_id	通道 ID	string	—
ctrl_info	备注	string	—

A.14.2.2 入参示例

```
{
    "query_para": {
        "op_type": "10600100",
```

```
            "app_no": "1060000",
            "key_id": "123456789",
            "st_id": "147258369",
        "ctrl_time": "1653365532",
        "ctrl_event":"1",
            "ctrl_type": "1",
            "ctrl_value": "1",
            "ctrl_machine": "jk-scd1",
            "ctrl_guarder_machine": "jk-scd1",
            "ctrl_user": "test001",
            "ctrl_guarder": "monitor001",
            "chn_id" : "1653365132",
            "ctrl_info": "default"
        }
    }
```

A.14.2.3 顺序预置出参键值说明：

参数	说明	数据类型	备注
ret_no	控制结果	String	详见错误代码
ret_msg	返回信息	String	—
app_no	应用号	String	—
key_id	控制目标 ID	String	—
ctrl_time	控制当前时间	String	unix 时间戳

A.14.2.4 出参示例：

```
{
    "query_result": {
        "ret_no": "1",
        "ret_msg": "control time out",
        "app_no": "1060000",
        "key_id": "123456789",
        "ctrl_time": "1653365532"
    }
}
```

A.14.3 顺控操作调用接口

应满足以下要求：

a） 接口原型：int jkmessage_invocation::messageSend(Message*msg,int length, JKMsg_destination *p=NULL);

b） 顺控操作调用采用消息总线交互模型，涉及操作票文件交互的流程部分，采用文件服务交互，服务端通过消息的形式告知客户端操作票文件路径和文件名，客户端通过文件服务接口获取。

c） 客户端与服务端消息交互报文为：

```
struct SEQ_CTRL_MESSAGE
{
    int         app_no;
    MLang::Long     ctrl_time;
    MLang::Long     st_id;
    MLang::Long     bay_id;
```

```
        MLang::STRING     tiket_name;
        MLang::STRING     tiket_path;
        MLang::STRING     ctrl_obj;
        int               step_no;
        char              ctrl_result;
        int               ret_no;
        MLang::STRING     ctrl_machine;
        MLang::STRING     ctrl_user;
        MLang::STRING     ctrl_guarder;
        MLang::STRING     ctrl_guarder_machine;
        MLang::STRING     msg;
        SEQ_CTRL_MESSAGE();
        SEQ_CTRL_MESSAGE(const SEQ_CTRL_MESSAGE&);
        SEQ_CTRL_MESSAGE&operator=(const SEQ_CTRL_MESSAGE&);
        void __write(MLang::OutputStream&__os);
        void __read(MLang::InputStream&__is);
    };
```

d） 客户端与服务端交互的消息通道见表 A.200；客户端与服务端交互的消息类型见表 A.201；

表 A.200 客户端与服务端交互消息通道

消息通道宏定义	备注
JK_API_MSGCHAN_S_TICKET_SEND	客户端->顺控服务
JK_API_MSGCHAN_S_TICKET_RECV	顺控服务->客户端

表 A.201 客户端与服务端交互的消息类型

消息类型宏定义	说明	备注
JK_API_MSGTYPE_S_TICKET_CTRL_CALL	召票	客户端->服务端
JK_API_MSGTYPE_S_TICKET_CTRL_CALL_CONFIRM	召票确认	服务端->客户端
JK_API_MSGTYPE_S_TICKET_CTRL_CALL_ANSWER_FILE	召票返回	服务端->客户端
JK_API_MSGTYPE_S_TICKET_CTRL_SIM	预演	客户端-服务端
JK_API_MSGTYPE_S_TICKET_CTRL_SIM_CONFIRM	预演确认	服务端->客户端
JK_API_MSGTYPE_S_TICKET_CTRL_SIM_ANSWER	预演返回结果	服务端->客户端
JK_API_MSGTYPE_S_TICKET_CTRL_EXEC	操作票执行	客户端-服务端
JK_API_MSGTYPE_S_TICKET_CTRL_EXEC_ANSWER	操作票执行确认	服务端->客户端
JK_API_MSGTYPE_S_TICKET_CTRL_EXEC_STEP	单步执行	客户端-服务端
JK_API_MSGTYPE_S_TICKET_CTRL_EXEC_STEP_ANSWER	操作票执行结果	服务端->客户端

JK_API_MSGTYPE_S_TICKET_CTRL_PAUSE	暂停	客户端-服务端
JK_API_MSGTYPE_S_TICKET_CTRL_PAUSE_ANSWER	暂停确认	服务端->客户端
JK_API_MSGTYPE_S_TICKET_CTRL_CONTINUE	继续	客户端-服务端
JK_API_MSGTYPE_S_TICKET_CTRL_CONTINUE_ANSWER	继续确认	服务端->客户端
JK_API_MSGTYPE_S_TICKET_CTRL_CANCEL	取消	客户端-服务端
JK_API_MSGTYPE_S_TICKET_CTRL_CANCEL_CONFIRM	取消确认	服务端->客户端
JK_API_MSGTYPE_S_TICKET_CTRL_STOP	终止	客户端-服务端
JK_API_MSGTYPE_S_TICKET_CTRL_STOP_CONFIRM	终止确认	服务端->客户端
JK_API_MSGTYPE_S_TICKET_CTRL_OVER	结束	客户端-服务端
JK_API_MSGTYPE_S_TICKET_CTRL_OVER_CONFIRM	结束确认	服务端->客户端
JK_API_MSGTYPE_S_TICKET_CALL_HEAD	召间隔所有票头	客户端-服务端
JK_API_MSGTYPE_S_TICKET_CALL_HEAD_ANSWER	召间隔所有票头返回	服务端->客户端
JK_API_MSGTYPE_S_TRANS_SWITCH_TICKET	下发倒闸操作票	客户端-服务端
JK_API_MSGTYPE_S_TRANS_SWITCH_TICKET_ANSWER	下发倒闸操作票返回	服务端->客户端
JK_API_MSGTYPE_S_TRANS_SWITCH_TICKET_FILE	倒闸操作票下发	客户端->服务端

A.14.4 参数定值修改服务接口

接口原型：参考 A.11 的服务调用示例代码

远方操作定值修改采用服务总线交互，包括定值召唤，修改定值预置，修改定值固化。

（1）定值召唤

a）客户端发送给服务端入参键值说明：

参数	类型	参数说明	是否必填
op_type	string	服务号	详见服务号代码
app_no	string	控制应用	*
ctrl_time	string	控制当前时间	*
st_id	string	控制目标厂站 ID	*
addr	string	103 地址	*
cpu_no	string	cpu 号	*
dz_area_no	string	对其进行召唤操作的定值区号	*
ctrl_machine	string	控制节点	*
ctrl_guarder_machine	string	监护节点	*
ctrl_user	string	控制用户	*
ctrl_guarder	string	监护用户	*

入参示例：

{

"op_type": "11300200",

"app_no": "1060000",

"ctrl_time": "123456789",

"st_id": "147258369",

"addr": "22",
"cpu_no":"1",
"dz_area_no": "1",
"ctrl_machine": "jk-scd1",
"ctrl_guarder_machine": "jk-scd1",
"ctrl_user": "test001",
"ctrl_guarder": "monitor001"
}

b）定值信息 DzVal 键值说明：

消息实体参数	类型	参数说明	是否必填
dz_name	string	定值名称	*
dz_val	string	定值实际值	*
dz_unit	string	定值量纲	*
dz_max_val	string	定值最大值	*
dz_min_val	string	定值最小值	*
int_digits	string	整数位	*
fract_digits	string	小数位	*
dz_step_val	string	步长	*
dz_type	string	定值类型: 0: 浮点型定值 1：整型定值 2：控制字定值 3：描述型定值	*
group_no	string	组号	*
entry_no	string	条目号	*
cpu_no	string	CPU 号	*

c）客户端接收服务端信息的键值说明：

消息实体参数	类型	参数说明	是否必填
app_no	string	控制应用	*
st_id	string	控制目标厂站 ID	*
addr	string	103 地址	*
cpu_no	string	cpu 号	*
dz_area_no	string	对其进行召唤操作的定值区号	*
vector<DzVal> &dz_val	vector<DzVal>	定值信息	*
ctrl_time	string	控制时间	*
ret_no	string	返回值：详见错误代码	*
ret_msg	string	返回信息	*

客户端接收服务端信息的键值示例：

{
"app_no": "1060000",
"st_id": "147258369",
"addr": "22",
"cpu_no":"1",
"dz_area_no": "1",
"dz_val": [{
"dz_name": "过流 I 段电流定值",
"dz_val": "2.3",
"dz_unit": "A",

```
            "dz_max_val": "99.99",
            "dz_min_val": "0",
            "int_digits":"4",
            "fract_digits": "2",
            "dz_step_val": "0.01",
            "dz_type": "0",
            "group_no": "2",
            "entry_no": "3",
            "cpu_no": "1"
        }],
        "ctrl_time": "123456789",
        "ret_no": "0",
        "ret_msg": "召唤定值成功"
    }
```


（2）定值修改预置、固化

a) 客户端发送给服务端信息的键值说明：

消息实体参数	类型	参数说明	是否必填
op_type	string	服务号	详见服务号代码
app_no	string	控制应用	*
ctrl_time	string	控制当前时间	*
st_id	string	控制目标厂站 ID	*
addr	string	103 地址	*
cpu_no	string	cpu 号	*
dz_area_no	string	对其进行召唤操作的定值区号	*
vector<RelayDzSeq> &dz_seq	vector<RelayDzSeq>	需要预置、固化的定值序列	*
ctrl_machine	string	控制节点	*
ctrl_guarder_machine	string	监护节点	*
ctrl_user	string	控制用户	*
ctrl_guarder	string	监护用户	*

定值预置、固化序列信息 RelayDzSeq 的键值说明：

消息实体参数	类型	参数说明	是否必填
group_no	string	召唤定值时返回的定值项组号	*
entry_no	string	召唤定值时返回的定值项条目号	*
cpu_no	string	召唤定值时返回的定值项 CPU 号	*
dz_type	string	定值类型: 0: 浮点型定值 1：整型定值 2：控制字定值 3：描述型定值	*
dz_modify_val	string	定值修改值	*

客户端发送给服务端信息的键值示例：


```
{
        "op_type": "10500200",
        "app_no": "1060000",
        "ctrl_time ": "123456789",
```

```
        "st_id": "147258369",
        "addr ": "22",
        "cpu_no ":"1",
        "dz_area_no ": "1",
        "dz_seq": [{
            "group_no": "2",
            "entry_no": "3",
            "cpu_no": "1",
            "dz_type": "0",
            "dz_modify_val": "2.34"
    }],
        "ctrl_machine": "jk-scd1",
        "ctrl_guarder_machine": "jk-scd1",
        "ctrl_user": "test001",
        "ctrl_guarder": "monitor001"
}
```


b) 客户端接收服务端信息的键值说明：

消息实体参数	类型	参数说明	是否必填
app_no	string	控制应用	*
st_id	string	控制目标厂站 ID	*
dev_id	string	控制目标设备 ID	*
dz_area_no	string	对其进行预置、固化操作的定值区号	*
ctrl_time	string	控制时间	*
ret_no	string	返回值：详见错误代码	*
ret_msg	string	返回信息	*

客户端接收服务端信息的键值示例：


```
{
        "app_no": "1060000",
        "st_id": "147258369",
        " dev_id ": "12233455667",
        "dz_area_no ": "1",
       "ctrl_time ": "123456789",
       "ret_no": "0",
       "ret_msg ": "修改定值成功"

}
```


A.14.5 二次设备服务

A.14.5.1 召唤状态量

A.14.4.1.1 接口原型：参考 A.11 服务调用示例代码。

A.14.4.1.2 入参键值说明见表 A.202：

表 A.202 召唤状态量入参说明

参数	说明	数据类型	备注
op_type	服务号	String	详见服务号代码
st_id	厂站 ID	String	—
relay_id	装置 ID	String	—

A.14.4.1.3 入参示例：

```
{
    "query_para": {
        "op_type": "11100200",
        "st_id": "01123100000001",
        "relay_id": "22250699010200013603"
    }
}
```

A.14.4.1.4 出参键值说明见表 A.203：

表 A.203 召唤状态量出参说明

参数	说明	数据类型	备注
ret_no	返回值	String	详见错误代码
ret_msg	返回信息	String	报错信息
data_list	数据列表	Array	—
yx_name	状态量名称	String	—
yx_val	状态量值	String	包括：分、合、不确定

A.14.4.1.5 出参示例：

```
{
    "query_result": {
        "ret_no":"0",
        "ret_msg": "",
        "data_list": [
        {
                "yx_name": "CPU03 零序方向 21 出口",
                "yx_val": "分"
        },
        {
                "yx_name": "CPU03 零序方向 21 出口",
                "yx_val": "分"
        }
        ]
    }
}
```

A.14.5.2 召唤模拟量

A.14.4.2.1 接口原型：参考 A.11 的服务调用示例代码。

A.14.4.2.2 入参键值说明见表 A.204：

表 A.204 召唤模拟量入参说明

参数	说明	数据类型	备注
op_type	服务号	String	详见服务号代码
st_id	厂站 ID	String	—
relay_id	装置 ID	String	—

A.14.4.2.3 入参示例：

```
{
    "query_para": {
        "op_type": "11200200",
        "st_id": "01123100000001",
        "relay_id": "22250699010200013603"
    }
```

```
}
```

A.14.4.2.4 出参键值说明见表 A.205：

表 A.205 召唤模拟量出参说明

参数	说明	数据类型	备注
ret_no	返回值	String	详见错误代码
ret_msg	返回信息	String	报错信息
data_list	数据列表	Array	—
yc_name	模拟量名称	String	—
yx_val	模拟量值	String	—

A.14.4.2.5 出参示例：

```
{
    "query_result": {
        "ret_no":"0",
        "ret_msg": "",
        "data_list": [{
                "yc_name": "装置温度",
                "yc_val": "49.3"
        },
        {
                "yc_name": "光强",
                "yc_val": "39.4"
        }
        ]
    }
}
```

A.14.5.3 召唤定值区号

A.14.4.3.1 接口原型：参考 A.11 的服务调用示例代码。

A.14.4.3.2 入参键值说明见表 A.206：

表 A.206 召唤定值区号入参说明

参数	说明	数据类型	备注
op_type	服务号	String	详见服务号代码
st_id	厂站 ID	String	—
relay_id	装置 ID	String	—

A.14.4.3.3 入参示例：

```
{
    "query_para": {
        "op_type": "11400200",
        "st_id": "01123100000001",
        "relay_id": "22250699010200013603"
    }
}
```

A.14.4.3.4 出参键值说明见表 A.207：

表 A.207 召唤定值区号出参说明

参数	说明	数据类型	备注
ret_no	返回值	String	详见错误代码
ret_msg	返回信息	String	报错信息
data_list	数据列表	Array	—
dz_area_name	定值区名称	String	—
dz_area_val	定值区号	String	—

A.14.4.3.5 出参示例：


```
{
    "query_result": {
        "ret_no":"0",
        "ret_msg": "",
        "data_list": [{
        "dz_area_name": "运行定值区号",
        "dz_area_val": "1"
        },
        ]
    }
}
```


A.14.5.4 召唤故障录波文件列表

A.14.4.4.1 接口原型：参考 A.11 的服务调用示例代码。

A.14.4.4.2 入参示例：


```
{
    "query_para": {
        "op_type": "21200200",
        "st_id": "01123100000001",
        "relay_id": "22250699010200013603",
        "start_time": "1640995200",
        "end_time": "1651363200"
    }
}
```


A.14.4.4.3 入参键值说明见表 A.208：

表 A.208 召唤故障录波文件列表入参说明

参数	说明	数据类型	备注
op_type	服务号	String	详见服务号代码
st_id	厂站 ID	String	—
relay_id	装置 ID	String	—
start_time	起始时间	String	unix 时间戳
end_time	终止时间	String	unix 时间戳

A.14.4.4.4 出参键值说明见表 A.209：

表 A.209 召唤故障录波文件列表出参说明

参数	说明	数据类型	备注
ret_no	返回值	String	详见错误代码
ret_msg	返回信息	String	报错信息
data_list	数据列表	Array	—
no	序号	String	—
time_str	录波文件时间字符串	String	—
file_name	文件名称	String	—
file_suffix	文件后缀	String	多个后缀用"/"分割
file_size	文件大小	String	单位：字节
file_path	存放路径	String	—
file_time	录波文件时间	String	unix 时间戳

A.14.4.4.5 出参示例：


```
{
    "query_result": {
```

```
            "ret_no":"0",
            "ret_msg": "",
        "data_list": [{
        "no": "1",
                    "time_str": "2019 年 01 月 05 日 16 时 28 分 32 秒 000",
                    "file_name": "PM2201A_PROT_918_20190105_161113_395",
                    "file_suffix": "cfg/dat/hdr",
                    "file_size": "183649",
                    "file_path": "var/relay/测试站/线路保护 1/201901/02",
                    "file_time": "193473057"
        },
        {
        "no": "2",
                    "time_str": "2019 年 01 月 05 日 16 时 28 分 32 秒 000",
                    "file_name": "PM2201A_PROT_918_20190105_161113_435",
                    "file_suffix": "cfg/dat/hdr",
                    "file_size": "183649",
                    "file_path": "var/relay/测试站/线路保护 1/201901/02",
                    "file_time": "193473057"
        }
        ]
        }
        }
```


A.14.5.5 召唤故障录波文件

A.14.4.5.1　接口原型：参考 A.11 的服务调用示例代码。

A.14.4.5.2　入参键值说明见表 A.210：

表 A.210 召唤故障录波文件入参说明

参数	说明	数据类型	备注
op_type	服务号	String	详见服务号代码
st_id	厂站 ID	String	—
relay_id	装置 ID	String	—
fault_file_time	录波文件时间	String	unix 时间戳
file_name	录波文件名称	String	—
file_suffix	文件后缀	String	多个后缀用"/"分割

A.14.4.5.3　入参示例：


```
{
    "query_para": {
        "op_type": "20300200",
        "st_id": "01123100000001",
            "relay_id": "22250699010200013603",
            "fault_file_time": "1640995200",
            "file_name": "PM2201A_PROT_918_20190105_161113_395",
            "file_suffix": "cfg/dat/hdr"
    }
}
```


A.14.4.5.4　出参键值说明见表 A.211：

表 A.211 召唤故障录波文件出参说明

参数	说明	数据类型	备注
ret_no	请求结果	String	详见错误代码
ret_msg	返回信息	String	报错信息
file_path	文件路径	String	文件服务主机内路径

A.14.4.5.5 出参示例：

```
{
        "query_result": {
        "ret_no": "0",
        "ret_msg": "",
        "file_path": "var/relay/测试站/线路保护 1/201901/02"
    }
}
```

A.14.5.6 召唤装置历史信息

A.14.4.6.1 接口原型：参考 A.11 的服务调用示例代码。

A.14.4.6.2 入参键值说明见表 A.212：

表 A.212 召唤装置历史信息入参说明

参数	说明	数据类型	备注
op_type	服务号	String	详见服务号代码
st_id	厂站 ID	String	—
relay_id	装置 ID	String	—
start_time	起始时间	String	unix 时间戳
end_time	终止时间	String	unix 时间戳

A.14.4.6.3 入参示例：

```
{
    "query_para": {
        "op_type": "11500200",
        "st_id": "01123100000001",
        "relay_id": "22250699010200013603",
        "start_time": "1640995200",
        "end_time": "1651363200"
    }
}
```

A.14.4.6.4 出参键值说明见表 A.213：

表 A.213 召唤装置历史信息出参说明

参数	说明	数据类型	备注
ret_no	返回值	String	详见错误代码
ret_msg	返回信息	String	报错信息
data_list	数据列表	Array	—
type	事件类型	String	—
time	时间字符串	String	—
name	时间详情	String	—
val	值描述	String	—

A.14.4.6.5 出参示例：

```
{
    "query_result": {
        "ret_no":"0",
        "ret_msg": "",
"data_list": [{
"type": "保护动作",
                "time": "2021 年 9 月 21 日 13 时 15 分 23 秒",
                "name": "距离一段动作",
                "val": "动作"
},
```

```
{
"type": "告警自检",
                    "time": "2021 年 9 月 21 日 13 时 15 分 23 秒",
                    "name": "告警自检 1",
                    "val": "告警消失"
}
]
    }
}
```

A.14.5.7 召唤通信信息

A.14.4.7.1 接口原型：参考 A.11 的服务调用示例代码

A.14.4.7.2 入参键值说明见表 A.214：

表 A.214 召唤通道信息入参说明

参数	说明	数据类型	备注
op_type	服务号	String	详见服务号代码
st_id	厂站 ID	String	—

A.14.4.7.3 入参示例：

```
{
    "query_para": {
        "op_type": "11600200",
        "st_id": "01123100000001"
    }
}
```

A.14.4.7.4 出参键值说明见表 A.215：

表 A.215 召唤通道信息出参说明

参数	说明	数据类型	备注
ret_no	返回值	String	详见错误代码
ret_msg	返回信息	String	报错信息
data_list	数据列表	Array	—
dev_name	装置名称	String	—
dev_val	装置状态	String	—

A.14.4.7.5 出参示例：

```
{
    "query_result": {
        "ret_no": "0",
        "ret_msg": "",
        "data_list": [{
                    "dev_name": "5115 线第二套主保护",
                    "dev_val": "正常"
            },
            {
                    "dev_name": "5115 线第一套主保护",
                    "dev_val": "中断"
            }
        ]
    }
}
```

A.14.5.8 召唤软压板

A.14.4.8.1　接口原型：参考 A.11 的服务调用示例代码。
A.14.4.8.2　入参键值说明见表 A.216：

表 A.216 召唤软压板入参说明

参数	说明	数据类型	备注
op_type	服务号	String	详见服务号代码
st_id	厂站 ID	String	—
relay_id	装置 ID	String	—

A.14.4.8.3　入参示例：

```
{
    "query_para": {
        "op_type": "11700200",
        "st_id": "01123100000001",
        "relay_id": "22250699010200013603"
    }
}
```

A.14.4.8.4　出参键值说明见表 A.217：

表 A.217 召唤软压板出参说明

参数	说明	数据类型	备注
ret_no	返回值	String	详见错误代码
ret_msg	返回信息	String	报错信息
data_list	数据列表	Array	—
yb_name	软压板名称	String	—
yb_val	软压板值	String	包括：分、合、不确定

A.14.4.8.5　出参示例：

```
{
    "query_result": {
        "ret_no": "0",
        "ret_msg": "",
        "data_list": [{
                "yb_name": "距离保护软压板",
                "yb_val": "分"
            },
            {
                "yb_name": "过流保护软压板",
                "yb_val": "合"
            }
        ]
    }
}
```

A.14.5.9 召唤定值

A.14.4.9.1　接口原型：参考 A.11 的服务调用示例代码。
A.14.4.9.2　此操作适用于 DL/T860 规约，入参键值说明见表 A.218：

表 A.218 召唤定值入参说明

参数	说明	数据类型	备注
op_type	服务号	String	详见服务号代码
st_id	厂站 ID	String	—
relay_id	装置 ID	String	—
zone	定值区号	String	—

A.14.4.9.3 入参示例：

```
{
    "query_para": {
        "op_type": "11300300",
        "st_id": "01123100000001",
        "relay_id": "12250699010200013603"
        "zone": "1"
    }
}
```

A.14.4.9.4 出参键值说明见表 A.219：

表 A.219 召唤定值出参说明

参数	说明	数据类型	备注
ret_no	返回值	String	详见错误代码
ret_msg	返回信息	String	报错信息
data_list	数据列表	Array	—
name	定值名称	String	—
type	定值类型	String	定值类型: 0：浮点型定值 1：整型定值 2：控制字定值 3：描述型定值
max_val	最大值	String	—
min_val	最小值	String	—
step_val	步长	String	—
dz_val	值	String	—
dimension	量纲	String	—
data_reference	数据引用	String	—

A.14.4.9.5 出参示例：

```
{
    "query_result": {
        "ret_no":"0",
        "ret_msg": "",
        data_list": [{
            "name": "变化量启动电流定值",
            "type": "0",
            "max_val": "99",
            "min_val": "0",
            "step_val": "0.01",
            "dz_val": "38.62",
            "dimension": "ohm",
            "data_reference": ""
          },
          {
            "name": "距离 I 段时间定值",
            "type": "1",
            "max_val": "50000",
            "min_val": "0",
            "step_val": "1",
            "dz_val": "50",
            "dimension": "ms",
            "data_reference": ""
          }
        ]
```

}
}

A.14.5.10 定值修改预置、固化

1) 客户端发送给服务端信息的键值说明见表 A.220；定值预置、固化序列信息 RelayDzSeq860 的键值说明见表 A.221；

表 A.220 客户端发送给服务端信息的键值说明

消息实体参数	类型	参数说明	是否必填
op_type	string	服务号	详见服务号代码
app_no	string	控制应用	*
ctrl_time	string	控制当前时间	*
st_id	string	控制目标厂站 ID	*
relay_id	string	装置 id	*
dz_area_no	string	定值区号	*
vector<RelayDzSeq860> &dz_seq	vector<RelayDzSeq860>	需要预置、固化的定值序列	*
ctrl_machine	string	控制节点	*
ctrl_guarder_machine	string	监护节点	*
ctrl_user	string	控制用户	*
ctrl_guarder	string	监护用户	*

表 A.221 定值预置、固化序列信息 RelayDzSeq860 的键值说明

消息实体参数	类型	参数说明	是否必填
data_reference	string	数据引用	*
dz_type	string	定值类型: 0: 浮点型定值 1：整型定值 2：控制字定值 3：描述型定值	*
dz_modify_val	string	定值修改值	*

2) 客户端发送给服务端信息的键值示例：

```
{
    "query_para": {
        "op_type": "11320200",
        "app_no": "1060000",
        "ctrl_time ": "123456789",
        "st_id": "147258369",
        "relay_id": "12250699010200013603",
        "dz_area_no": "1",
        "dz_seq": [{
            “data_reference”:””;
            "dz_type": "0",
            "dz_modify_val": "2.34"
        }],
        "ctrl_machine": "jk-scd1",
        "ctrl_guarder_machine": "jk-scd1",
        "ctrl_user": "test001",
        "ctrl_guarder": "monitor001"
```

```
        }
    }
```

3) 客户端接收服务端信息的键值说明见表 A.222：

表 A.222 客户端接收服务端信息的键值说明

消息实体参数	类型	参数说明	是否必填
app_no	string	控制应用	*
st_id	string	控制目标厂站 ID	*
relay_id	string	控制目标设备 ID	*
dz_area_no	string	对其进行预置、固化操作的定值区号	*
ctrl_time	string	控制时间	*
ret_no	string	返回值：详见错误代码	*
ret_msg	string	返回信息	*

4) 客户端接收服务端信息的键值示例：

```
{
    "query_result": {
        "app_no": "1060000",
        "st_id": "147258369",
        "relay_id": "12233455667",
        "dz_area_no": "1",
        "ctrl_time": "123456789",
        "ret_no": "0",
        "ret_msg": "修改定值成功"
    }
}
```

A.14.6 其他服务

A.14.6.1 下发一般文件

A.14.5.1.1 接口原型：参考 A.11 的服务调用示例代码。

A.14.5.1.2 入参键值说明见表 A.223：

表 A.223 下发一般文件入参说明

参数	说明	数据类型	备注
op_type	服务号	String	详见服务号代码
st_id	厂站 ID	String	—
form	文件下发形式	String	0 文件内容 1 文件在文件服务主机的路径
file_serv_path	文件在文件服务主机的路径	String	form 为 1 时生效
file_content	Base64 编码后的文件内容	String	form 为 0 时生效
file_content_len	Base64 编码后的字节长度	String	form 为 0 时生效
file_path	下发至变电站综合应用主机的路径	String	—
file_name	文件名称	String	—
ied_name	ied 名称	String	—

A.14.5.1.3 入参示例：


```
{
    "query_para": {
        "op_type": "20100100",
        "st_id": "01123100000001",
        "form": "0",
        "file_serv_path": "var/his",
        "file_content": "aGVsbG8=",
        "file_content_len": "8",
        "file_path": "/HISDATA",
        "file_name": "HISDATA_analog_query_14.res",
        "ied_name": "PL2201A"
    }
}
```


A.14.5.1.4 出参键值说明见表 A.224：

表 A.224 下发一般文件出参说明

参数	说明	数据类型	备注
ret_no	返回结果	String	详见错误代码
ret_msg	返回信息	String	—

A.14.5.1.5 出参示例：


```
{
    "query_result": {
        "ret_no": "-1",
        "ret_msg": "file_serv_path cannot be recognised"
    }
}
```


A.14.6.2 召唤一般文件

A.14.5.2.1 接口原型：参考 A.11 的服务调用示例代码。

A.14.5.2.2 入参键值说明见表 A.225：

表 A.225 召唤一般文件入参说明

参数	说明	数据类型	备注
op_type	服务号	String	详见服务号代码
st_id	厂站 ID	String	—
form	文件返回形式	String	0 文件内容 1 文件在文件服务主机的路径
file_serv_path	文件在文件服务主机的路径	String	form 为 1 时生效
file_path	下发至变电站综合应用主机的路径	String	—
file_name	文件名称	String	—
ied_name	ied 名称	String	—

A.14.5.2.3 入参示例：


```
{
    "query_para": {
        "op_type": "20200100",
        "st_id": "01123100000001",
        "form": "0",
        "file_serv_path": "var/his",
```

```
            "file_path": "/HISDATA",
            "file_name": "HISDATA_analog_query_14.res",
            "ied_name": "PL2201A"
        }
    }
```

A.14.5.2.4 出参键值说明见表 A.226：

表 A.226 召唤一般文件出参说明

参数	说明	数据类型	备注
ret_no	返回结果	String	详见错误代码
ret_msg	返回信息	String	—
form	文件返回形式	String	0 文件内容 1 文件在文件服务主机的路径
file_serv_path	文件在文件服务主机的路径	String	form 为 1 时生效
file_content	Base64 编码后的文件内容	String	form 为 0 时生效
file_content_len	Base64 编码后的字节长度	String	form 为 0 时生效

A.14.5.2.5 出参示例：

```
{
    "query_result": {
        "ret_no": "0",
        "ret_msg": "",
        "form": "0",
        "file_srv_path": "var/relay/测试站/线路保护 1/201901/02",
        "file_content": "aGVsbG8=",
        "file_content_len": "8"
    }
}
```

A.14.6.3 调阅文件列表

A.14.5.3.1 接口原型：参考 A.11 的服务调用示例代码。

A.14.5.3.2 入参键值说明见表 A.227：

表 A.227 调阅文件列表入参说明

参数	说明	数据类型	备注
op_type	服务号	String	详见服务号代码
st_id	厂站 ID	String	—
start_time	列表起始时间	String	unix 时间戳
end_time	列表结束时间	String	unix 时间戳
file_path	文件访问路径	String	详见《变电站二次系统站控系统技术规范 第 3 部分 综合应用主机技术规范》分册 A.1 DL/T860 文件服务访问路径
ied_name	ied 名称	String	—

A.14.5.3.3 入参示例：

```
{
    "query_para": {
        "op_type": "21100100",
        "st_id": "01123100000001",
```

```
            "start_time": "1640995200",
            "end_time": "1651363200",
            "file_path": "/COMMAND",
            "ied_name": "PL2201A"
        }
    }
```

A.14.5.3.4 出参键值说明见表 A.228：

表 A.228 调阅文件列表出参说明

参数	说明	数据类型	备注
ret_no	返回结果	String	详见错误代码
ret_msg	返回信息	String	—
num	文件数量	String	如果 ret_no 和 num 同时为 0，表示请求成功但目录下没有文件
file_list	文件列表	Array	—
file_name	文件名称	String	—
file_time	文件修改时间	String	—

A.14.5.3.5 出参示例：

```
{
    "query_result": {
        "ret_no": "0",
        "ret_msg": "",
        "num": "3",
        "file_list": [{
                "file_name": "substation.scd",
                "file_time": "2022-02-03 19:89:39"
            },
            {
                "file_name": "substation1.zip",
                "file_time": "2022-02-03 19:89:39"
            },
            {
                "file_name": "iedname_configured.cid",
                "file_time": "2022-02-03 19:89:39"
            }
        ]
    }
}
```

A.14.6.4 RPC 模型调阅

A.14.5.4.1 接口原型：参考 A.11 的服务调用示例代码。

A.14.5.4.2 入参键值说明见表 A.229：

表 A.229 RPC 模型调阅入参说明

参数	说明	数据类型	备注
op_type	服务号	String	详见服务号代码
st_id	厂站 ID	String	—

A.14.5.4.3 入参示例：

```
{
    "query_para": {
        "op_type": "11800000",
```

```
        "st_id": "01123100000001"
    }
}
```

A.14.5.4.4　出参键值说明见表 A.230：

表 A.230 RPC 模型调阅出参说明

参数	说明	数据类型	备注
ret_no	返回结果	String	详见错误代码
ret_msg	返回信息	String	—
num	数据数量	String	如果 ret_no 和 num 同时为 0，表示请求成功但没有数据
data_list	数据列表	Array	—
name	调阅类型	String	参照数据通信网关机规范 G2
type	数据类型	String	参照数据通信网关机规范 G2
ndesc	调阅类型描述	String	参照数据通信网关机规范 G2
tdesc	数据类型描述	String	参照数据通信网关机规范 G2

A.14.5.4.5 出参示例：

```
{
    "query_result": {
        "ret_no": "0",
        "ret_msg": "",
        "num": "2",
        "data_list": [{
                "name": "ALLYC",
                "type": "YC",
                "ndesc": "所有遥测",
                "tdesc": "遥测"
            },
            {
                "name": "ALLYX",
                "type": "YX",
                "ndesc": "所有遥信",
                "tdesc": "遥信"
            }
        ]
    }
}
```

A.14.6.5 RPC 数据调阅

A.14.5.5.1　接口原型：参考 A.11 的服务调用示例代码。

A.14.5.5.2　入参键值说明见表 A.231：

表 A.231 RPC 数据调阅入参说明

参数	说明	数据类型	备注
op_type	服务号	String	详见服务号代码
st_id	厂站 ID	String	—
command	请求命令	String	参照数据通信网关机规

			范 G2

A.14.5.5.3 入参示例：

```
{
    "query_para": {
        "op_type": "11900100",
        "st_id": "01123100000001",
        "command": "PL2201|PL2202.ALM"
    }
}
```

A.14.5.5.4 出参键值说明见表 A.232：

表 A.232 RPC 数据调阅出参说明

参数	说明	数据类型	备注
ret_no	返回结果	String	详见错误代码
ret_msg	返回信息	String	—
num	数据数量	String	如果 ret_no 和 num 同时为 0，表示请求成功但没有数据
data_list	数据列表	Array	—
no	编号	String	—
ref	DL/T860 参引	String	—
desc	测点描述	String	—
call_type	调阅类型	String	—
value	测点值	String	—
q	测点品质	String	—

A.14.5.5.5 出参示例：

```
{
    "query_result": {
        "ret_no": "0",
        "ret_msg": "",
        "num": "2",
        "data_list": [{
                "no": "1",
                "ref": "ABCD",
                "desc": "500kV 测试站 5012 开关有功值",
                "call_type": "全部遥测",
                "value": "20",
                "q": "4096"
            },
            {
                "no": "2",
                "ref": "ABCD",
                "desc": "500kV 测试站 5012 开关有功值",
                "call_type": "全部遥测",
                "value": "20",
                "q": "4096"
            }
        ]
    }
}
```

A.14.6.6 读数据集

A.14.5.6.1 接口原型：参考 A.11 的服务调用示例代码。

A.14.5.6.2 入参键值说明见表 A.233：

表 A.233 读数据集入参说明

参数	说明	数据类型	备注
op_type	请求类型	String	详见服务号代码
st_id	厂站 id	String	—
			—
datasetReference	数据集名称	String	—
referenceAfter	数据集值	String	为空表示读数据集所有值，不为空表示读指定点之后的值

A.14.5.6.3 入参示例：

```
{
  "query_para": {
    "op_type": "11030300",
    "st_id": "113997366155018324"
    "datasetReference": "COMM/LLN0.dsPnt500KV",
    "referenceAfter":"PROT/MMXU1.A.phsa,PROT/MMXU1.a.phsB....."
  }
}
```

A.14.5.6.4 出参键值说明见表 A.234：

表 A.234 读数据集出参说明

参数	说明	数据类型	备注
ret_no	返回结果	String	详见错误代码
ret_msg	返回信息	String	—
num	数据数量	String	—
data_list	应答数据	Array	—
value	数据值	String	—
q	数据品质	String	—
data_reference	数据参引	String	—

A.14.5.6.5 出参示例：

```
{
  "query_result": {
  "ret_no": "0",
  "ret_msg": "",
  "num": "2",
  "data_list": [{
    "value": "20",
    "q": "4096",
    "data_reference":"PROT/MMXU1.A.phsA"
  },
  {
    "value": "20",
    "q": "4096",
    "data_reference":"PROT/MMXU1.A.phsB"
  }
  ]
}
}
```

A.14.6.7 创建/删除动态集

A.14.6.6.1 接口原型：参考 A.11 的服务调用示例代码。

A.14.6.6.2 入参键值说明见表 A.235：

表 A.235 设置/删除动态数据集入参说明

参数	说明	数据类型	备注
op_type	请求类型	string	详见服务号代码
st_id	厂站 id	string	—
dataset_reference	动态数据集名称	string	—
datalist	测点列表	array	创建数据集测点列表
data_reference	测点数据参引	string	创建数据集测点
fc	功能约束	string	—

A.14.6.6.3 入参示例：

```
{
  "query_para": {
    "op_type": "11040300/11050300",
    "st_id": "113997366155018324"
    "dataset_reference": "dsDinTemp",
     "data_list": [{
                    "data_reference":"P_L2211BLD0/GGIO1.ST.Ind1",
                   "fc":"ST",
                },
                {
                    "data_reference":"P_L2211BLD0/GGIO1.ST.Ind2",
                   "fc":"ST",
                 }
    ]
  }
}
```

A.14.6.6.4 出参键值说明见表 A.236：

表 A.236 设置/删除动态数据集入参说明

参数	说明	数据类型	备注
ret_no	返回结果	String	详见错误代码
ret_msg	返回信息	String	—

A.14.6.6.5 出参示例：

```
{
    "query_result": {
        "ret_no": "-1",
        "ret_msg": "create dynamic dataset error"
    }
}
```

A.14.7 错误码

错误码见表A.237：

表 A.237 错误码

错误码	错误信息
0	成功
1	失败
11	服务请求超时

13	参数不合法
20	目标处于控制中状态
21	目标不处于控制中状态
22	目标被禁止控制
23	目标状态与遥控的状态相同
1002	连接认证服务器失败
1004	应用认证信息加密失败
1005	应用认证信息解密失败
1006	应用认证结果加密失败
1007	应用认证结果解密失败
1202	Token 超时失效
1203	Token 验签失败
1206	未找到 token
1301	时间戳超时
1302	nonce 已存在
1303	安全报文签名验证失败
-1	未知错误

A.15 实时数据获取接口

实时数据服务接口采用消息方式进行数据发布，平台与应用间约定消息体和消息类型，应用可通过平台消息总线（A.5）接收指定的消息来获取实时数据的变化情况。状态量与模拟量公用同一个消息结构，状态量填写 char 数据结构，模拟量填写 float 数据结构，同时以消息事件号进行区分：

a） 主设备实时数据消息通道：JK_API_MSGCHAN_S_MAIN_REAL_DATA；

b） 辅设备实时数据消息通道：JK_API_MSGCHAN_S_AUX_REAL_DATA；

c） 主设备实时数据消息类型：JK_API_MSGTYPE_S_MAIN_CHANGE_DATA；

d） 辅设备实时数据消息类型：JK_API_MSGTYPE_S_AUX_CHANGE_DATA。

```
struct ChangePkgHead
{
    int    char_num;
    int    int_num;
    int    float_num;
    int    udata_num;
    int    table_num;
    int    status_num;
    void __write(MLang::OutputStream&__os)const;
    void __read(MLang::InputStream&__is);
};
struct CCharStru
{
    int    app_no;
    TKeyID  keyid;
    char    value;
    int    status;
    void __write(MLang::OutputStream&__os)const;
    void __read(MLang::InputStream&__is);
};
struct CIntStru
{
    int    app_no;
    TKeyID  keyid;
    int    value;
    int    status;
    void __write(MLang::OutputStream&__os)const;
```

```
        void __read(MLang::InputStream&__is);
    };
    struct CFloatStru
    {
        int    app_no;
        TKeyID  keyid;
        float   value;
        int    status;
        void __write(MLang::OutputStream&__os)const;
        void __read(MLang::InputStream&__is);
    };
    struct CTableStru
    {
        int    app_no;
        int    table_no;
        void __write(MLang::OutputStream&__os)const;
        void __read(MLang::InputStream&__is);
    };
    struct CStatusStru
    {
        int    app_no;
        MLang::Long    key;
        int    status;
        int    status_column_id;
        void __write(MLang::OutputStream&__os)const;
        void __read(MLang::InputStream&__is);
    };
    typedef MLang::VECTOR<CCharStru> SeqCCharStru;
    typedef MLang::VECTOR<CIntStru> SeqCIntStru;
    typedef MLang::VECTOR<CFloatStru> SeqCFloatStru;
    typedef MLang::VECTOR<CUdataStru> SeqCUdataStru;
    typedef MLang::VECTOR<CTableStru> SeqCTableStru;
    typedef MLang::VECTOR<CStatusStru> SeqCStatusStru;
    struct ChangeDataPkg
    {
        ChangePkgHead   package_head;
        SeqCCharStru    char_info;
        SeqCIntStru     int_info;
        SeqCFloatStru   float_info;
        SeqCUdataStru   udata_info;
        SeqCTableStru   table_info;
        SeqCStatusStru  status_info;
        ChangeDataPkg();
        ChangeDataPkg(const ChangeDataPkg&);
        ChangeDataPkg&operator=(const ChangeDataPkg&);
        void __write(MLang::OutputStream&__os)const;
        void __read(MLang::InputStream&__is);
};
```

A.16 告警窗右键菜单扩展接口（QT）

A.16.1 接口原型：QStringList showName() const;

A.16.2 接口说明：返回菜单的显示名称，见表 A.238：

表 A.238 告警窗右键菜单扩展接口返回菜单说明

接口参数	类型	输入/输出	参数说明	是否必填
返回值	QStringList	Out	插件名称	

A.16.3 接口原型：QWidget *createPluginWidget(vector<WarnDataInfo> &warnInfos,QWidget *parent,QString menuName,QString userName,long respValue);

A.16.4 接口说明：传入告警数据 warnInfos 和告警窗口指针*parent，返回插件界面窗口指针，见表 A.239：

表 A.239 告警窗右键菜单扩展接口参数说明

接口参数	类型	输入/输出	参数说明	是否必填
warnInfos	Vector&	In	告警窗口数据	是
*parent	QWidget*	In	告警窗口指针	是
QString	menuName	In	菜单名称	是
QString	userName	In	用户名称	是
long	respValue	In	当前节点登录的责任区信息	是
返回值	QWidget *	Out	插件窗口指针	—

A.16.5 结构描述：

```
typedef struct warn_data_struct
{
    short type; //告警类型
    int status; //告警状态
    long occur_time; //告警时间
    int customized_group; //告警所属等级
    QString warn_str; //告警字符串
    QString warn_key_str; //告警关键字字符串
    long key_id;        //告警 key
    long fac_id;        //告警厂站 ID
    long resp_id;       //责任区 ID
    long bay_id;        //间隔 ID
    long vlty_id;         //电压 ID
    unsigned char if_need_fg_warn; //告警是否需要复归
    unsigned char fg_value;         //告警复归状态
    unsigned char confirm_status;  //确认状态
}WarnDataInfo
```

A.16.5 处理过程及返回值说明：当某个插件类窗口需要通过告警窗右键菜单调用，需要通过 showName()接口告知告警窗此插件的名称，通过*createPluginWidget 接口传入告警窗数据，返回此插件窗口指针。

A.16.6 调用示例：

```
QStringList TestOne::showName() const
{
    return QStringList() << tr("事件详情");
}
QWidget *TestOne::createPluginWidget(vector<WarnDataInfo> &warnInfos,QWidget *parent,QString menuName,QString userName,long respValue)
{
    PluginWidget *pluginWidget = new PluginWidget(warnInfos,parent,menuName,userName,respValue); //插件界面类
    return pluginWidget;
```

}

A.17 告警窗右键菜单扩展接口（JAVA）

A.17.1 接口原型：void registerPopupMenu(String className, String methodName, String menuName);
A.17.2 接口包名：alarmrealtime.popupMenu；
A.17.3 接口类名：AlarmPopupMenuOperation；
A.17.4 接口说明：返回菜单的显示名称，见表 A.240：

表 A.240 告警窗右键菜单扩展 JAVA 接口返回菜单说明

接口参数	类型	输入/输出	参数说明	是否必填
className	String	In	右键菜单类名（包含包名）	是
methodName	String	In	右键菜单执行操作方法名（方法没有参数）	是
menuName	String	In	右键菜单名	是

A.17.5 接口原型：Alarm_Socut_TableDataBean getAlarmBean();
A.17.6 接口包名：alarmdata.bean；
A.17.7 接口类名：Alarm_Socut_TableDataBean；
A.17.8 接口说明：返回告警窗右键点击告警对象:，见表 A.241：

表 A.241 告警窗右键菜单扩展接口入参说明

接口参数	类型	输入/输出	参数说明	是否必填
返回值	Alarm_Socut_TableDataBean	Out	—	—

A.17.9 处理过程及返回值说明：当需要通过告警窗右键菜单调用自定义类执行方法时，首先通过 registerPopupMenu(String className, String methodName, String menuName)接口告知告警窗此菜单类的信息，通过 getAlarmBean()接口可获取告警窗右键点击告警对象，可在自定义右键菜单类 methodName 方法实现点击菜单操作。

A.17.10 调用示例：

```
AlarmPopupMenuOperation.registerPopupMenu(“popupMenu.menus.WarnConfirm”, “oper”, “告警确认”);
public void oper() {
 Alarm_Socut_TableDataBean bean = AlarmPopupMenuOperation. getAlarmBean();
 confirmAlarm(bean);
}
```

A.17.11 接口 AlarmPopupMenuOperation.getAlarmBean()返回告警对象 Alarm_Socut_TableDataBean，下表列出 Alarm_Socut_TableDataBean 属性说明，见表 A.242：

表 A.242 告警返回参数说明

属性名	说明
type	告警类型
warnContext	告警内容
warnTime	告警发生时间
facId	告警所属厂站 ID
bayId	告警所属间隔 ID
status	告警状态

confirmStatus	确认状态
keyId	告警所属设备 ID
reserved_1	告警等级
vityId	电压 ID
if_fg	是否复归

A.18 安全Ⅳ区JAVA接口

A.18.1 查询库识别信息（DBID）(JAVA)

应满足以下要求：

a） 接口原型：public static String RTDBOpen(String domain,String contextName, String appName,String ip)；

b） 接口包名：jk.service；

c） 接口类名：IDataService；

d） 接口说明：DBID 是实时库查询接口的参数之一，进行实时库数据查询前，需要通过下列方法取得；

e） 参数列表见表 A.243：

表 A.243 RTDBOpen 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
domain	String	In	域名称	是
contextName	String	In	态名称	是
appName	String	In	应用名称	是
ip	String	In	主机名称(若为 null，则参加资源定位)	是
返回值	String	Out	库识别信息	—

f） 接口示例（RTDBOpen 接口访问）：

```
String dbid = IDataService.RTDBOpen("local","realtime", "scada", null);
```

A.18.2 查询全表数据(JAVA)

A.18.2.1 接口原型：public static Vector<Vector<DataUnit>> RTDBRead(String DBid, String tableName)throws MidLayerAccessException；

A.18.2.2 接口包名：jk.service；

A.18.2.3 接口类名：IMiddataService；

A.18.2.4 接口返回类型数据单元包名：jk.agency.message.base；

A.18.2.5 接口返回类型数据单元类名：DataUnit；

A.18.2.6 异常类包名：jk.services.agency；

A.18.2.7 异常类名：MidLayerAccessException；

A.18.2.8 接口说明：查询指定表的全部数据；

A.18.2.9 参数列表见表 A.244：

表 A.244 RTDBRead 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
DBid	String	In	DBID	是
tableName	String	In	表名	是
返回值	Vector<Vector<DataUnit>>	Out	表数据向量	—

A.18.2.10 接口示例（RTDBRead 接口访问）：

```
String dbid = IDataService.RTDBOpen("local","realtime", "scada", null);
```

```
try {
Vector<Vector<DataUnit>> vec = IMiddataService.RTDBRead(dbid, 表名);
} catch (MidLayerAccessException e) {
e.printStackTrace();
}
```

A.18.3 按照单关键字查询记录(JAVA)

A.18.3.1 接口原型：public static Vector<DataUnit> RTDBRead(String DBid, String tableName, String key) throws MidLayerAccessException；
A.18.3.2 接口包名：jk.service；
A.18.3.3 接口类名：IMiddataService；
A.18.3.4 接口返回类型数据单元包名：jk.agency.message.base；
A.18.3.5 接口返回类型数据单元类名：DataUnit；
A.18.3.6 异常类包名：jk.services.agency；
A.18.3.7 异常类名：MidLayerAccessException；
A.18.3.8 接口说明：根据关键字查询指定表数据；
A.18.3.9 参数列表见表 A.245：

表 A.245 RTDBRead 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
DBid	String	In	DBID	是
tableName	String	In	表名	是
key	String	In	关键字	是
返回值	Vector<Vector<DataUnit>>	Out	表数据向量	—

A.18.3.10 接口示例（RTDBRead 接口访问）：

```
String dbid = IDataService.RTDBOpen("local","realtime", "scada", null);
try {
Vector<DataUnit> vec = IMiddataService.RTDBRead(dbid, 表名,KEYID);
} catch (MidLayerAccessException e) {
e.printStackTrace();
}
```

A.18.4 删除记录(JAVA)

A.18.4.1 接口原型：public static void RTDBRemove(String DBid, String tableName, String key) throws MidLayerA ccessException；
A.18.4.2 接口包名：jk.service；
A.18.4.3 接口类名：IMiddataService；
A.18.4.4 异常类包名：jk.services.agency；
A.18.4.5 异常类名：MidLayerAccessException；
A.18.4.6 接口说明：根据关键字删除表指定数据；
A.18.4.7 参数列表见表 A.246：

表 A.246 RTDBRemove 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
DBid	String	In	DBID	是
tableName	String	In	表名	是
key	String	In	关键字	是
返回值	无	Out	无	—

A.18.4.8　接口示例（RTDBRemove 接口访问）：

```
String dbid = IDataService.RTDBOpen("local","realtime", "scada", null);
try {
IMiddataService.RTDBRemove(dbid , 表名, 关键字);
} catch (MidLayerAccessException e) {
printStackTrace();
}
```

A.18.5 插入记录(JAVA)

A.18.5.1 接口原型：public static void RTDBInsert(String DBid, String tableName, Vector data)throws MidLayerAccessException；

A.18.5.2 接口包名：jk.service；

A.18.5.3 接口类名：IMiddataService；

A.18.5.4 异常类包名：jk.services.agency；

A.18.5.5 异常类名：MidLayerAccessException；

A.18.5.6 接口说明：要插入的多行数据的三级向量，最内层的向量存放域名和域值对儿；中间一层存放若干个最内层的向量，代表一条记录的多个域；最外层代表多条记录；

A.18.5.7 参数列表见表 A.247：

表 A.247 RTDBInsert 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
DBid	String	In	DBID	是
tableName	String	In	表名	是
data	Vector	In	要插入的多行数据的三级向量	是
返回值	无	Out	无	—

A.18.5.8 接口示例（RTDBInsert 接口访问）：

```
String dbid = IDataService.RTDBOpen("local","realtime", "scada", null);
Vector allData = new Vector();
Vector singleData1 = new Vector();

Vector beanData1 = new Vector();
beanData1.add(0, 域名 1);
beanData1.add(1, 域值 a1);
singleData1.add(beanData1);

Vector beanData2= new Vector();
beanData2.add(0, 域名 2);
beanData2.add(1, 域值 a2);
singleData1.add(beanData2);

allData.add(singleData1);

Vector singleData2 = new Vector();

Vector beanData2 = new Vector();
beanData2.add(0, 域名 1);
beanData2.add(1, 域值 b1);
singleData2.add(beanData2);

Vector beanData2= new Vector();
beanData2.add(0, 域名 2);
```

```
beanData2.add(1, 域值 b2);
singleData2.add(beanData2);

allData.add(singleData2);

try {
IMiddataService.RTDBInsert(dbid, 表名, allData);
} catch (MidLayerAccessException e) {
e.printStackTrace();
}
```

A.18.6 更新记录(JAVA)

A.18.6.1 接口原型：public static void RTDBUpdate(String DBid, String tableName, String key, Vector<String> property, Vector data) throws MidLayerAccessException；
A.18.6.2 接口包名：jk.service；
A.18.6.3 接口类名：IMiddataService
A.18.6.4 异常类包名：jk.services.agency；
A.18.6.5 异常类名：MidLayerAccessException；
A.18.6.6 接口说明：更新指定单关键字值的一条记录；
A.18.6.7 参数列表见表 A.248：

表 A.248 RTDBUpdate 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
DBid	String	In	DBID	是
tableName	String	In	表名	是
key	String	In	关键字	是
property	Vector<String>	In	需要更新的域名	是
data	Vector	In	需要更新的域值	是
返回值	无	Out	无	—

A.18.6.8 接口示例（RTDBUpdate 接口访问）：

```
String dbid = IDataService.RTDBOpen("local","realtime", "scada", null);
Vector column= new Vector();
column.add(域名 1);
column.add(域名 2);
Vector value= new Vector();
value.add(域值 1);
value.add(域值 2);
try {
IMiddataService.RTDBUpdate(dbid,表名 关键字, column, value);
} catch (MidLayerAccessException e1) {
e1.printStackTrace();
}
```

A.18.7 SQL服务接口(JAVA)

A.18.7.1 接口原型：public static Vector<Vector<DataUnit>> RDBCommand(String sql) throws MidLayerAccessException；
A.18.7.2 接口包名：jk.service；
A.18.7.3 接口类名：IMidhsService；
A.18.7.4 接口返回类型数据单元包名：jk.agency.message.base；
A.18.7.5 接口返回类型数据单元类名：DataUnit；

A.18.7.6 异常类包名：jk.services.agency；
A.18.7.7 异常类名：MidLayerAccessException；
A.18.7.8 接口说明：调用 midhs 服务接口，执行 SQL 语句；
A.18.7.9 参数列表见表 A.249：

表 A.249 RDBCommand 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
sql	String	In	SQL 语句	是
返回值	Vector<Vector<Data Unit>>	Out	数据结果	—

A.18.7.10 接口示例（RDBCommand 接口访问）：

```
String sql = "select * from tableName where column=value'";
Vector<Vector<DataUnit>> vec = IMidhsService.RDBCommand(sql);
```

A.18.8 获取文件java接口

A.18.8.1 接口原型：public static byte[] getByteFile(String domain, String fileName) throws DataFaultException, ServiceConnectionException,LocatorException, ProxyFaultException；
A.18.8.2 接口包名：jk.service；
A.18.8.3 接口类名：IFileservService；
A.18.8.4 异常类包名：jk.services.agency.exception；
A.18.8.5 异常类名：DataFaultException、ServiceConnectionException、LocatorException、ProxyFaultException；
A.18.8.6 接口说明：获取指定区域文件；
A.18.8.7 参数列表见表 A.250：

表 A.250 getByteFile 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
domain	String	In	区域名称,变电站为"substation"	是
fileName	String	In	文件全名	是
返回值	byte[]	Out	表数据向量	—

A.18.8.8 接口示例（getByteFile 接口访问）：

```
byte[] temp = IFileservService.getByteFile("local",文件名);
byte[] temp = IFileservService.getByteFile("substation",文件名);
```

A.18.9 权限验证

A.18.9.1 接口原型：public static synchronized boolean isUserValid(String user_name, String password) throws Exception ；
A.18.9.2 接口包名：jk.service；
A.18.9.3 接口类名：IPermService；
A.18.9.4 接口说明：根据用户名与密码校验用户；
A.18.9.5 参数列表见表 A.251：

表 A.251 isUserValid 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
user_name	String	In	用户名	是
password	String	In	用户密码	是
返回值	boolean	Out	数据结果	—

A.18.9.6 接口示例（isUserValid 接口访问）：

```
try {
```

```
boolean result= IPermService.isUserValid(用户名,密码);
System.out.println(result);
} catch (Exception e) {
e.printStackTrace();
}
```

返回值说明
true 用户校验成功
false 用户校验失败

A.19 人机扩展接口（JAVA）

人机其他应用右键以插件形式进行拓展，应用通过调用” 添加右键菜单项”接口实现右键菜单的拓展。

A.19.1 添加右键菜单项

A.19.1.1 接口原型：public static void addPopupMenu(String className)；
A.19.1.2 接口包名：jk.pluginterface；
A.19.1.3 接口类名：PluginInterface；
A.19.1.4 接口说明：在现有菜单基础上，添加一个右键菜单项；
A.19.1.5 参数列表见表 A.252：

表 A.252 addPopupMenu 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
className	String	In	右键菜单事件实现类名称	是
返回值	无	Out	—	—

A.19.1.6 接口示例（addPopupMenu 接口访问）：

```
String className = “菜单实现类名”;
PluginInterface.addPopupMenu(className );
```

A.19.2 记录日志信息接口

A.19.2.1 接口原型 public static void logInfo(Class classObject,Object message)；
A.19.2.2 接口包名：jk.pluginterface；
A.19.2.3 接口类名：PluginInterface；
A.19.2.4 参数列表见表 A.253：

表 A.253 logInfo 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
classObject	Class	In	调用日志接口的类对象	是
message	Object	In	日志信息	是
返回值	无	Out	—	—

A.19.2.5 接口示例（logInfo 接口访问）：

```
PluginInterface.logInfo(this.getClass(), message);
```

A.19.3 嵌入swing界面

A.19.3.1 接口原型 pluginterface.ui.PlugInPanel；
A.19.3.2 接口包名：jk.pluginterface.ui；

A.19.3.3　接口类名：PlugInPanel；
A.19.3.4　接口说明：第三方应用的嵌入面板需要继承 PlugInPanel，并实现此类的抽象方法；
A.19.3.5　参数列表见表 A.254：

表 A.254 PlugInPanel 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
返回值	无	Out	—	—

A.19.3.6　接口示例（PlugInPanel 接口访问）：

```
public class DemoPanel extends PlugInPanel{
}
```

A.19.4 浏览器切换态接口

A.19.4.1　接口原型 public static void setCurrentContext(Stringcontext)；
A.19.4.2　接口包名：jk.pluginterface；
A.19.4.3　接口类名：PluginInterface；
A.19.4.4　接口说明： 设置浏览器态属性；
A.19.4.5　参数列表见表 A.255：

表 A.255 setCurrentContext 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
context	String	in	态信息	是
返回值	无	Out	—	—

A.19.4.6　接口示例（setCurrentContext 接口访问）：

```
PluginInterface.setCurrentContext(“study”);
```

A.19.5 获取图形对应的URL接口

A.19.5.1　接口原型 public static String getURL(String graphName)；
A.19.5.2　接口包名：jk.pluginterface.tool；
A.19.5.3　接口类名：URLTool；
A.19.5.4　接口说明： 根据画面名称获取 url 地址；
A.19.5.5　参数列表见表 A.256：

表 A.256 getURL 接口参数表

接口参数	类型	输入输出	参数说明	是否必填
graphName	String	in	画面名称	是
返回值	String	Out	url 地址	—

A.19.5.6　接口示例（getURL 接口访问）：

```
String url= URLTool.getURL(画面名称);
```

A.20　人机扩展接口（QT）

图形与应用主要是以弹出右键菜单的方式实现与应用的互动，Callback是图形与应用交互的一个基本功能，各应用都需要实现自己的Callback，以下各API接口由应用实现具体方法。

A.17.1—A.17.4 包含头文件：jk_G_IAppCallback.h,各独立应用继承该头文件的 IAppCallback 动态库名称格式：libDLL_xxxxxx.so，存放路径~/lib/graph/。配置文件 app_callback.ini，内容：

```
[app_config]
app_count = 2  #总的应用数
app_id_1=100000 # 应用 1（应用号 100000）
app_id_2=400000# 应用 2（应用号 400000）
```

```
    [100000] #应用 1 对应的动态库信息
    callback_count=1
    callback_dll_name_1 = libDLL_ExampleCallback.so

    [400000] #应用 2 对应的动态库信息
    callback_count=2
    callback_dll_name_1 = libDLL_Example1Callback.so
    callback_dll_name_2 = libDLL_Example2Callback.so
```

a）图元信息结构：

```
class DLL_CLASS JKObjInfo
{
public:
    JKObjInfo();
    JKObjInfo(const JKObjInfo &src);
    JKObjInfo& operator=(const JKObjInfo &src);
    ~JKObjInfo();

    void CreatePrivateInfo(int obj_priv_type);

public:
    int         obj_type; // 图元类型；
    char        is_on_pin;// 是否在端子上，默认为-1；
    int         obj_flag; // 图元的属性int          obj_id;   // 图形ID号；
    int         obj_priv_type;      // 特殊数据类型，可以是数据，标志牌，列表，曲线；
    void*       obj_priv_info;      // 特殊数据信息，可以是数据，标志牌，列表，曲线；
    JKObjDynInfoVec  sub_obj_dyn_info_vec;
};

typedef std::vector<JKObjInfo> JKObjInfoVec;

class DLL_CLASS JKObjDynInfo
{
public:
    JKObjDynInfo();
    JKObjDynInfo(const JKObjDynInfo &src);
    ~JKObjDynInfo();
    JKObjDynInfo &operator=(const JKObjDynInfo &src);

public:
    int             app_id;             // 应用号；
    int             table_id;           // 表号，默认为-1；
    short           value_column_id;    // 值域域号    ----- 联库时连接的域，默认为-1；
    short           state_column_id;    // 状态域域号  ----- 由值域决定，默认为-1；
    char            key_type;           // 关键字的数据类型，默认为-1；
    short           key_len;            // 关键字的长度，默认为8；
    CGBuffer        key_value;          // 关键字的内容；
    int64           vol_type;       // 电压等级；
    int             report_type;  //动态属性的状态，例如 REALTIME_DYNAMIC_REPORT
    time_t          start_time;     // 历史值：历史时间，（report_type=HISTORY_DYNAMIC_REPORT）
```

```
                                    // 统计值：统计的开始时间；
    time_t          end_time;       // 统计值：统计的结束时间；（report_type=
STATISTIC_DYNAMIC_REPORT）
    char            statistic_type; // 统计类型：数值统计/开关/越限/工况
    int             interval;       // 间隔（当统计类型==数值统计时使用）
    int         menu_type; // 类型选择--选项来自菜单表（当统计类型=开关/越限/ 工况/）
    char        statistic_mode; // 取值类型：最大值/最小值/平均值

};
typedef std::vector<JKObjDynInfo> JKObjDynInfoVec;
```

b）图形信息：

```
class DLL_CLASS JKCurGraphInfo
{
public:
   JKCurGraphInfo();

public:
    CGString graph_name;        //图形名称;
    int   serial_no;      //图形编号;
    int    cursor_pos_x;
   int    cursor_pos_y;
   int    cursor_gpos_x;
   int    cursor_gpos_y;
   time_t  cur_time;
   int    cur_app_id;            //当前图形应用号;
    short      cur_context_no;          //当前图形态;
   int64   graph_fac_id; //图形所属厂站id
    int   user_id;        //当前画面操作人ID号;
    CGString user_name;
    long node_id; //图形所在机器的节点号
    CGString display_name;//当前窗口的节点ip和屏幕号信息，如localhost:0.0
    char graph_status;        //图形的状态;
    char app_type;      //APP_NORMAL, APP_OPTICKET;
    char topo_policy;//着色策略，如正常着色（COLOR_STYLE_NORMAL）
    int   wnd_id;
};
```

c）菜单信息：

```
class DLL_CLASS JKMenuInfo
{
public:
   JKMenuInfo();
   ~JKMenuInfo();
    JKMenuInfo(const JKMenuInfo &src);
    JKMenuInfo &operator=(const JKMenuInfo &src);

public:
    CGString menu_name;//显示的菜单名称
    CGString menu_icon_name;

   int            menu_type;//菜单类型，例如 NORMAL_MENU
   int            menu_state;//菜单状态，文字纵向显示，为1；其他，为0
```

```
    CGString    font_color; //例 "black"，"darkblue"，"gray55"
    bool        is_italic;//是否斜体
    bool        is_blackland;//是否粗体
    bool        is_underground;//是否带下划线
    bool        is_enable;//菜单项是否使能
    int         align_type; //对齐方式 如Qt::AlignCenter
    CGString    op_accel; //操作快捷键
      JKOpInfoVec  op_info_vec;
    JKMenuInfoVec sub_menu_vec;
};
typedef std::vector< JKMenuInfo* > JKMenuInfoVec;

      d）操作信息：
class DLL_CLASS JKOpInfo
{
public:
      JKOpInfo();

public:
      unsigned int    interface_type; //SIMPLE_OP -- 简单图元的操作
                                      //GRAPH_OP  -- 图形操作
      int     op_id;;// 菜单所对应的功能号，用于区分不同的操作，每个操作对象唯一，由平台统一分配操作功能号段
      CGString graph_name;          //打开图形的名称;
      bool showing_plane[50];//打开图形的显示平面;
      int    op_app_id;// 该操作对应的应用号
    int64 op_record_id;//操作设备ID
      short       op_column_id;             //值域号;
      bool cal_before;            //显示计算前;
      bool cal_biaoyao;           //显示标幺值;

      JKCurveOpInfo curve_op_info;

};
typedef std::vector< JKOpInfo > JKOpInfoVec;

class DLL_CLASS JKBaseCurveOpInfo
{
public:
      JKBaseCurveOpInfo();
public:
      int64       app_key_id;//测点id
      short             dot_number;             //该曲线小数位数；－1为图形指定，大于等于零有效
      bool        need_set_start_time;
      //datetimecurve_start_time;    //CurveOp只需要该值；
      CGString domain;              //域名；
      CGString    curve_name;       //曲线名称
};
class DLL_CLASS JKCurveOpInfo
{
public:
      JKCurveOpInfo();
```

```
public:
    CGString curve_menu_name;//指定曲线工具的曲线菜单
    int          curve_time_span;   //以秒为单位
    JKBaseCurveOpInfoVecbase_curve_op_info_vec;//右键曲线配置定义
};
typedef std::vector< JKBaseCurveOpInfo > JKBaseCurveOpInfoVec;
```

说明：设备类的右键操作，如信息检索、控制等，JKObjInfo ::obj_type，JKObjInfo ::obj_id按实际填写, 有动态属性的设备，JKObjInfo::sub_obj_dyn_info_vec 不能为空，其中，JKObjDynInfo的app_id，table_id，value_column_id，state_column_id，key_type，key_len，key_value按实际情况赋值，其他属性根据应用场景赋值。

对于右键操作，还需提供图形的相关信息给应用，如JKCurGraphInfo的graph_name，user_id，user_name，如果画面有所属厂站信息，还需提供graph_fac_id，其他属性根据应用场景赋值。

应用根据设备信息和图形信息返回相应的菜单信息，菜单信息根据实际情况赋值JKMenuInfo的menu_name，menu_type，is_enable，op_info_vec等，其他属性根据应用场景赋值；JKOpInfo的interface_type，op_id按实际情况赋值，其他属性根据应用场景赋值。

基于QtPlugin方式的右键菜单，返回参数不需要对op_info_vec赋值，插件根据menu_name执行相应操作。

A.20.1 按应用扩展对象的右键菜单

virtual HRESULT GetMenu(const JKObjInfoVec & obj_dyn_info_vec, const JKCurGraphInfo & graph_info, JKMenuInfoVec & menu_info_vec)=0;

a) 【接口说明】该接口为应用 Callback 回调接口，用于获取设备、画面对象右键菜单。。

b) 【主要参数】见表 A.257：

表 A.257 按应用扩展对象的右键菜单

接口参数	类型	输入输出	参数（返回值）说明	备注
obj_dyn_info_vec	JKObjInfoVec	In	图元信息	obj_dyn_info_vec 的 size 为零时表示画面空白处右键操作
graph_info	JKCurGraphInfo	In	图形信息	—
menu_info_vec	JKMenuInfoVec	Out	对象菜单内容	—

c) 【返回值】-1：获取对象菜单失败；0：获取对象菜单成功

d) 【调用示例】

```
Int CAGCCallback::GetMenu(const JKObjInfoVec & obj_info_vec, const JKCurGraphInfo & graph_info, JKMenuInfoVec & menu_info_vec)
{
    char section_name[40]={0};
    strcpy(section_name,"区域数据菜单");
    OpInfo op_info;
    op_info.interface_type  = GRAPH_OP;
    op_info.op_id        = GRAPH_OP_CHANGE_PIC;
    op_info.op_record_id   = opa_id;
    op_info.graph_name    = "xxx.agc.pic.g";
    menu_info_vec[0]->op_info_vec.push_back(op_info);
    return 1;
}
```

A.20.2 按应用获取对象信息提示

virtual HRESULT GetTip(const JKObjInfoVec & obj_dyn_info_vec, const JKCurGraphInfo & graph_info,QString & tip)=0;

a）【接口说明】该接口为应用 Callback 回调接口，用于获取设备信息提示。

b）【主要参数】见表 A.258：

表 A.258 按应用获取对象信息提示

接口参数	类型	输入输出	参数（返回值）说明	备注
obj_dyn_info_vec	JKObjInfoVec	IN	图元信息	—
graph_info	JKCurGraphInfo	IN	图形信息	—
tip	QString	OUT	对象信息	—

c）【返回值】-1：获取设备信息失败；0：获取设备信息成功。

d）【调用示例】

```
IDrawObj *obj = obj_list.Object();
int sub_objid = obj->GetObjId();
QString info_tip
int sub_ret = GetDataAccess()->GetSubTip(sub_graph_name,sub_objid,info_tip);
if(sub_ret < 0)
{
      return sub_ret;
}
else
{
   gDebug("GetSubTip sub_ret=%d",sub_ret);
   return sub_ret;
}
```

A.20.3 设置应用号域态号

virtual HRESULT SetAppNo(int app_id, short context_no,QString domain=″″)=0;

a）【接口说明】该接口为应用 Callback 回调接口，用于设置应用号(态号，域名信息)。

b）【主要参数】见表 A.259：

表 A.259 设置应用号域态号

接口参数	类型	输入输出	参数（返回值）说明	备注
app_id	int	IN	应用号	—
context_no	short	IN	态号	—
domain	QString	IN	域名信息	—

c）【返回值】-1：设置应用号(态号，域名信息)失败；0：设置应用号(态号，域名信息)成功。

d）【调用示例】

```
HRESULT CAGCCallback::SetAppNo(int app_id, short context_no,QString domain)
{
      m_AppID = app_id;
      m_CtxNo = context_no;
   m_ Domain = domain;
      return GRAPH_NO_ERROR;
};
```

A.20.4 对当前设备操作

virtual HRESULT OpCallback(const JKObjInfoVec & obj_dyn_info_vec, const JKCurGraphInfo & graph_info, const int op_id, QWidget * parent,int event_type=-1)=0;

a）【接口说明】该接口为应用 Callback 回调接口，用于简单设备操作接口，如鼠标单击，双击操作。

b）【主要参数】见表 A.260：

表 A.260 对当前设备操作

接口参数	类型	输入输出	参数（返回值）说明	备注
obj_dyn_info_vec	JKObjInfoVec	IN	图元信息	—
graph_info	JKCurGraphInfo	IN	图形信息	—
op_id	int	IN	操作号	—
parent	QWidget	IN	父窗体	—
event_type	int	IN	事件类型	—

c）【返回值】-1：设备操作失败；0：设备操作成功。

d）【调用示例】

```
HRESULT CAGCCallback::OpCallback(const JKObjInfoVec & obj_info_vec, const JKCurGraphInfo & graph_info, const int op_id, QWidget * parent,int event_type)
{
    if ( event_type!=-1 ) {
        return GRAPH_ERROR;
    }
}
```

A.20.5 按插件方式扩展对象右键功能

A.20.5.1 加载基于 QtPlugin 开发的插件，图形将设备动态信息和当前图形信息发送给插件。插件根据这些信息判断是否提供对当前设备的操作，如果有，回传给图形。用户点击插件菜单后，调用插件的 Action()接口，将选中菜单的菜单名传给插件对象，由插件对象执行后续操作。按插件方式扩展对象右键功能接口参数见表 A.261。

A.20.5.2 动态库存放路径~/lib/graph/plugins。

virtual HRESULT GetMenu(const JKObjInfoVec & obj_dyn_info_vec, const JKCurGraphInfo & graph_info, JKMenuInfoVec & menu_info_vec) = 0；

表 A.261 按插件方式扩展对象右键功能

接口参数	类型	输入输出	参数（返回值）说明	备注
obj_dyn_info_vec	JKObjInfoVec	In	图元信息	—
graph_info	JKCurGraphInfo	In	图形信息	—
menu_info_vec	JKMenuInfoVec	Out	对象菜单内容	—

【返回值】-1：获取插件右键菜单失败；0：

A.20.5.3 获取插件右键菜单成功，见表 A.262。

virtual void Action(QString menu_name, QWidget * parent) = 0;

说明：右键菜单对应用的功能（槽）函数。

表 A.262 右键菜单对应用的功能（槽）函数

接口参数	类型	输入输出	参数（返回值）说明	备注
menu_name	QString	In	菜单名	—
parent	QWidget	In	父窗体	—

【返回值】无。

virtual QString Name() const = 0;//返回插件名称。

【返回值】返回插件名称，类型 QString。

A.20.6 按插件方式扩展集成图元

A.20.6.1 集成图元是一类用于特定场景的综合展示组件，行为上类似于普通图元，可以在编辑器中编辑，关联模型，并在浏览器中展示。以下系列接口支撑应用在图形编辑器中以扩展集成图元方式加入图元库中，并支持在浏览器中的显示。动态库名称格式：libJKDLL_xxxxxx.so，存放路径~/lib/graph/applib。

```
virtual   void   SetPropertyValue(QString  property_name, QVariant value)=0;
```

A.20.6.2 【接口说明】设置图元属性，用于编辑器中集成图元属性设置，见表 A.263：

表 A.263 按插件方式扩展集成图元

接口参数	类型	输入输出	参数（返回值）说明	备注
property_name	QString	In	属性名	—
value	QVariant	In	值	—

【调用示例】

```
void  CTestObj:: SetPropertyValue(QString  property_name, QVariant value)
{
   if (property_name == QObject::tr("文本背景色").toLocal8Bit().data())
   {
      m_app_background = value.asInt();
      QPalette palette = text->palette();
      palette.setColor(QPalette::Base, m_app_background);
      text->setPalette(palette);
   }
}
```

```
virtual   void   GetPropertyList(QList< CustomPropertyInfo >  &property_list)=0;
```

A.20.6.3 【接口说明】获取图元属性设置列表，用于编辑器属性的显示，见表 A.264：

表 A.264 获取图元属性设置列表

接口参数	类型	输入输出	参数（返回值）说明	备注
property_list	QList< CustomPropertyInfo >	OUT	属性列表	—

【调用示例】

```
typedef struct custom_property_struct
{
  QString   property_name;//属性名称
  QString   tip;          //提示
  int       show_type;    //图元属性的显示类型
                    // 0:int;1double;2:bool;3:uint;4:color;6:stringlist7:text;8:linestyle
  QVariant  value;
  int       property_style;
  QVariant  cur_item;
  custom_property_struct()
      {
        property_name=QString("");
        tip= QString("");
        show_type=0;
```

```
            value= QString("");
            property_style = 1;
            cur_item = QString("");
        }


    }CustomPropertyInfo;
    void  CTestObj:: GetPropertyList(QList< CustomPropertyInfo >  &property_list)
    {
        CustomPropertyInfo property;
        property.property_name = QObject::tr("文本背景色");
        property.show_type =4;
        property.tip = QObject::tr("文本背景色");
        property.value = QVariant(m_app_background);
        property_list.append(property);
    }

    virtual     void SetGeometry(const QRect& rect)=0;
```

A.20.6.4【接口说明】设置位置，见表 A.265：

表 A.265 设置位置

接口参数	类型	输入输出	参数（返回值）说明	备注
rect	QRect	In	图元位置	—

【调用示例】

```
    void CTestObjWrap::SetGeometry(const QRect& rect)
    {
        QWidget::setGeometry(rect);
    }

    virtual       intWrite(QDomElement &dom_obj)=0;
```

A.20.6.5【接口说明】写 G 文件，见表 A.266：

表 A.266 写 G 文件接口

接口参数	类型	输入输出	参数（返回值）说明	备注
dom_obj	QDomElement	In	图元信息	—

【调用示例】

```
    int CTestObj::Write(QDomElement &dom_obj)
    {
        dom_obj.setAttribute("app_background", QString::number(m_app_background)); //将背景色写入 g 文件
        return 0;
    }

    virtual       intRead(QDomElement &dom_obj)=0;
```

A.20.6.6【接口说明】读 G 文件，见表 A.267：

表 A.267 读 G 文件接口

接口参数	类型	输入输出	参数（返回值）说明	备注
dom_obj	QDomElement	Out	图元信息	—

【调用示例】

```
int CTestObj::Read(QDomElement &dom_obj)
{
   m_app_background = dom_obj.attribute("app_background").toInt();
   QPalette palette = text->palette();
   palette.setColor(QPalette::Base, m_app_background);
   text->setPalette(palette);
   return 0;
}
```

```
virtual        int     ReleaseObj()=0;
```

A.20.6.7【接口说明】资源释放。

【返回值】1：释放成功，-1：失败。

【调用示例】

```
int  CTestOb::ReleaseObj()
{
   close();
   return 1;
}
```

```
virtual        void    ContextChanged(int appNo,int contextNo)=0;
```

A.20.6.8【接口说明】集成图元的态号，应用号设置，见表 A.268：

表 A.268 集成图元态号、应用号设置

接口参数	类型	输入输出	参数（返回值）说明	备注
appNo	int	In	应用号	—
contextNo	int	In	态号	—

【调用示例】

```
void CTestObj::ContextChanged(int appNo, int contextNo)
{
   m_appNo = appNo;
   m_contextNo = contextNo;
}
```

```
virtual        void    PopProperty()=0;
```

A.20.6.9【接口说明】弹出集成图元的属性对话框调用

【调用示例】

```
int    CTestObj:: PopProperty()
{
   Extension * dlg = new Extension(this);
   dlg->show();
      return 0;
}
```

```
virtual        void    GetDynamicInfoOfSearch(std::vector<DynamicInfoOfSearch> &vDynInfo)=0;
```

A.20.6.10【接口说明】获取从 search 拖拽的动态信息，见表 A.269：

表 A.269 获取动态信息

接口参数	类型	输入输出	参数（返回值）说明	备注
vDynInfo	std::vector<Dyn	Out	模型信息	—

	amicInfoOfSearch>			

【调用示例】

```
typedef struct _DynamicInfoOfSearch
{
    int          app_id;      //应用号
    int          table_id;    //表号
    short        column_id;   //值域域号
    short        column_type; //域值类型
    short        column_len;  //域值长度
    short        state_column_id;  //状态域域号，对应state_column_id，联库时根据图元类型指定
    CGUByte      key_type;//关键字类型
    short        key_len; //关键字长度
    CGBuffer   key_value; //关键字内容
  int64        vol_type; //电压等级
  CGString   domain;
}
void CTestObj::GetDynamicInfoOfSearch(std::vector<DynamicInfoOfSearch>
                                      &vDynInfo)
{
  vDynInfo = m_DyInfoVec;
}
```

```
virtual      void   SetDynamicInfoOfSearch(std::vector<DynamicInfoOfSearch> vDynInfo)=0;
```

A.20.6.11【接口说明】设置从 search 拖拽的动态信息，见表 A.270：

表 A.270 拖拽的动态信息

接口参数	类型	输入输出	参数（返回值）说明	备注
vDynInfo	std::vector<DynamicInfoOfSearch>	In	模型信息	—

【调用示例】

```
void CTestObj::SetDynamicInfoOfSearch(std::vector<DynamicInfoOfSearch>
                                      vDynInfo)
{
  m_DyInfoVec = vDynInfo;
}
```

```
virtual      void   SetData(int group, int num, double val) = 0;
```

A.20.6.12【接口说明】直接设置动态对象数值，动态对象的动态数据可以分多组。预留扩展，暂未使用，见表 A.271：

表 A.271 设置动态对象数值

接口参数	类型	输入输出	参数（返回值）说明	备注
group	int	In	动态数据组数	—
num	int	In	每组动态数据个数	—
val	double	In	动态数据数值	—

【调用示例】

N.A.

```
virtual      void   CopyShape(IAppWidget *ptrWidget) = 0;
```

A.20.6.13【接口说明】复制集成图元时，集成图元对象如有特殊参数或属性，可通过此接口复制，

见表 A.272：

表 A.272 复制集成图元

接口参数	类型	输入输出	参数（返回值）说明	备注
ptrWidget	IAppWidget *	In	待拷贝对象的指针	—

【调用示例】

```
IAppWidget * src_widget;
IAppWidget * widget;
widget-> CopyShape(src_widget);
```

```
virtual void ReceiveMessage(const AppMessageStru &message)
```

A.20.6.14【接口说明】人机程序通过此接口向集成图元发送消息，见表 A.273：

表 A.273 人机程序通过此接口向集成图元发送消息

接口参数	类型	输入输出	参数（返回值）说明	备注
message	AppMessageStru	In	消息内容	—

【调用示例】

```
IAppWidget * widget;
AppMessageStru msg;
widget-> ReceiveMessage (msg);
```

```
virtual int CreateAppWidget(IAppWidget** pp_obj, QWidget * m_parent, QString method, GraphInfoInf graphInfo, IAppMessageHandle *IAppMessageHandle) = 0;
```

A.20.6.15【接口说明】IAppWidgetFactory 工厂类方法，用于创建继承图元对象，见表 A.274：

表 A.274 IAppWidgetFactory 工厂类方法

接口参数	类型	输入输出	参数（返回值）说明	备注
pp_obj	IAppWidget**	Out	返回新创建的集成图元指针	—
m_parent	QWidget *	In	父窗口指针	—
method	QString	In	方法名	—
graphInfo	GraphInfoInf	In	图形信息	—
IAppMessageHandle	IAppMessageHandle *	In	用于和图形通信的 MessageHandle 指针，扩展用	—

【返回值】0：创建集成图元成功，-1：失败。

【调用示例】

```
JK_API::IAppWidgetFactory *cfac;
IAppWidget* m_pDllObj_jk;
JK_API::GraphInfoInf gInfo;
gInfo.app_id = 10000; //示例参数
gInfo.contextNo = 1; //示例参数
gInfo.env_type = 1;//示例参数
gInfo.graph_name = QObject::tr(graphName.c_str());

QString appPara = GetAppPara();
if (!appPara.isEmpty())
{
```

```
        gInfo.app_para = appPara;
    }
    gInfo.domain_name = m_pGraphEnv->GetDomainName().c_str();
    QString method_name = “test”;
    JK_API::IAppMessageHandle *msghandle = NULL;
    int ret = cfac->CreateAppWidget(&m_pDllObj_jk, m_pParent, method_name, gInfo,
msghandle);
```

```
virtual int     Release() = 0;
```

A.20.6.16【接口说明】IAppWidgetFactory 工厂类方法，用于释放工厂类对象。

【返回值】0：释放成功，-1：失败。

【调用示例】

```
        JK_API::IAppWidgetFactory *cfac;
cfac->Releas();
virtual void       GetExternMethods(std::map<QString, QString> &m_methods) = 0;
```

A.20.6.17【接口说明】IAppWidgetFactory 工厂类方法，返回集成图元对象的所有方法和方法描述，见表 A.275：

表 A.275 IAppWidgetFactory 工厂类方法返回集成图元对象

接口参数	类型	输入输出	参数（返回值）说明	备注
m_methods	std::map<QString, QString> &	Out	一个方法对应一个集成图元工厂类所创建的一个集成图元对象；方法描述用于编辑器显示	—

```
virtual CGString    GetDescription() = 0;
```

A.20.6.18 【接口说明】IAppWidgetFactory 工厂类方法，返回各应用动态库的中文描述.扩展用。

【返回值】类型 CGString，动态库的描述。

```
virtual int       ReleaseAppWidget(IAppWidget** pp_obj) = 0;
```

A.20.6.19 【接口说明】IAppWidgetFactory 工厂类方法，用于释放集成图元对象，见表 A.276：

表 A.276 IAppWidgetFactory 工厂类方法释放集成图元对象

接口参数	类型	输入输出	参数（返回值）说明	备注
pp_obj	IAppWidget**	In	集成图元对象指针	—

【返回值】0：释放成功，-1：失败。

A.20.7 带应用态信息打开画面

A.20.7.1 图形浏览器工具提供带应用号态号跳转和打开图形画面，支撑应用按需调阅系统图形。

A.20.7.2 调用方法：命令行参数调用。

跳转：GExplorer–pic 文件名称 –console

打开：GExplorer –change –pic 文件名称 –console

切换态：GExplorer –context 态号 –pic 文件名称 –console

切换应用：GExplorer -appid 应用号 –pic 文件名称 –console

A.20.7.2 调用示例：

示例 1：GExplorer -pic 测试站.fac.pic.g -console

示例 2：GExplorer –change -pic 测试站.fac.pic.g -console

示例 3：GExplorer –context -5 测试站.fac.pic.g -console

示例 4：GExplorer -appid 100000 测试站.fac.pic.g -console

A.21　安全区IV浏览器获取图形接口

A.21.1 获取图形接口

B/S 图形提供图形浏览接口 url，通过 graph 图形名称入参的方式，可在浏览器中打开变电站接线图及间隔分图，图形 url 入参示例如下：

http://ip:host/.../.../xxx.html?graph=xx 变电站接线图.sys.pic.g;

A.21.2 集成组件展示框架

集成组件展示框架支持应用集成组件扩展，可通过配置文件、接口等方式实现扩展。配置文件内容可包括：组件在 CIM/G 文件中的标签名称、脚本名称、调用的程序（可包括嵌入的 html 或 js 等）名称等。接口可用于创建 div，并将组件的属性、div 的 id 等信息传递给组件，组件接收到属性、div 的 id 等信息后，创建组件图元并获取数据，最终 div 在 html 中展示。配置文件的内容、接口定义仅作示例参考，不做强制要求。

A.21.3 右键菜单扩展接口

右键菜单扩展接口提供通用的跨域解决方案，支持使用配置文件的方式对不同应用、不同设备图元进行右键菜单的扩展，通过接口的方式进行数据交互。

右键菜单配置文件中配置菜单名、方法、过滤条件（用于区分不同的应用、设备）、应用入口函数名称、应用所需的入口参数（包括设备的 keyid、G 文件中的 id、G 文件中组件标签名）。

A.22　自动对点

自动对点服务接口采用异步消息方式进行交互，平台与应用间约定消息体和消息类型，根据业务交互过程，支撑应用完成功能实现。自动对点相关宏定义见表 A.277：

表 A.277 自动对点相关宏定义

分类	类型	宏名
消息通道	消息通道	JK_API_MSGCHAN_S_AUTOACCEPTANCE
	控制通道	JK_API_MSGCHAN_S_AUTOACCEPTANCE_CTRL
下行	自动对点遥信事件	JK_API_MSGTYPE_S_ACPT_YX_SETTING
	自动对点遥测事件	JK_API_MSGTYPE_S_ACPT_YC_SETTING
	自动对点遥控事件	JK_API_MSGTYPE_S_ACPT_YK_SETTING
	自动对点遥调事件	JK_API_MSGTYPE_S_ACPT_YT_SETTING
	自动对点设点事件	JK_API_MSGTYPE_S_ACPT_SET_SETTING
	自动对点遥控预置	JK_API_MSGTYPE_S_ACPT_YK_PREV_SETTING
	自动对点遥控撤销	JK_API_MSGTYPE_S_ACPT_YK_PREV_CANCEL
	自动对点遥控执行	JK_API_MSGTYPE_S_ACPT_YK_EXEC_SETTING
	自动对点调档预置	JK_API_MSGTYPE_S_ACPT_TAP_PREV_SETTING
	自动对点调档撤销	JK_API_MSGTYPE_S_ACPT_TAP_PREV_CANCEL
	自动对点调档执行	JK_API_MSGTYPE_S_ACPT_TAP_EXEC_SETTING
	自动对点设点执行	JK_API_MSGTYPE_S_ACPT_SET_EXEC_SETTING
上行	自动对点变化遥信	JK_API_MSGTYPE_S_ACPT_MT_YX_CHANGE
	自动对点变化遥测	JK_API_MSGTYPE_S_ACPT_MT_YC_CHANGE
	自动对点 SOE	JK_API_MSGTYPE_S_ACPT_MT_YX_SOE
	自动对点遥控预置应答	JK_API_MSGTYPE_S_ACPT_MT_YK_PREV_REPLY
	自动对点遥控撤销应答	JK_API_MSGTYPE_S_ACPT_MT_YK_EXEC_REPLY
	自动对点调档预置应答	JK_API_MSGTYPE_S_ACPT_MT_TAP_PREV_REPLY
	自动对点调档执行应答	JK_API_MSGTYPE_S_ACPT_MT_TAP_EXEC_REPLY
	自动对点调档撤销应答	JK_API_MSGTYPE_S_ACPT_MT_TAP_CANCEL_REPLY
	自动对点设点执行应答	JK_API_MSGTYPE_S_ACPT_MT_SET_REPLY

A.22.1 应用向前置的消息交互定义

消息结构如下：

```
struct ScadaAcptPkgHead
{
   int        data_num;
   MLang::Long        second;
   short      msecond;
   short      flag_value;
   int        para;
   void __write(MLang::OutputStream& __os)const;
   void __read(MLang::InputStream& __is);
};
typedef MLang::VECTOR<MLang::Long> ChannelId_SEQ;
struct SetAcptFlagPkg_XP
{
   ScadaAcptPkgHead              package_head;
   ChannelId_SEQ     channel_seq;
   int        time;
   MLang::Long        keyid;
   int        ctrl_value;
   float      set_value;
   int        reserved1;
   int        reserved2;
   MLang::STRING   filename;
   MLang::STRING   path;
   SetAcptFlagPkg_XP();
   SetAcptFlagPkg_XP(const SetAcptFlagPkg_XP&);
   SetAcptFlagPkg_XP& operator=(const SetAcptFlagPkg_XP&);
   void __write(MLang::OutputStream& __os)const;
   void __read(MLang::InputStream& __is);
};
```

A.22.2 前置向应用消息交互定义

```
struct FesPkgNewHead
{
   int        package_type;
   int        data_num;
   MLang::Long        second;
```

```
    short     msecond;
    int       para;
    void __write(MLang::OutputStream& __os)const;
    void __read(MLang::InputStream& __is);
};
struct ChangeYxNew
{
    MLang::Long       chan_id;
    MLang::Long       keyid;
    char      value;
    int       status;
    MLang::Long       second;
    short     msecond;
    int       reserved1;
    int       reserved2;
    void __write(MLang::OutputStream& __os)const;
    void __read(MLang::InputStream& __is);
};
typedef MLang::VECTOR<ChangeYxNew> ChangeYx_SEQ_New;
struct ChangeYxPkgNew_XP
{
    FesPkgNewHead   package_head;
    ChangeYx_SEQ_New          yx_seq;
    ChangeYxPkgNew_XP();
    ChangeYxPkgNew_XP(const ChangeYxPkgNew_XP&);
    ChangeYxPkgNew_XP& operator=(const ChangeYxPkgNew_XP&);
    void __write(MLang::OutputStream& __os)const;
    void __read(MLang::InputStream& __is);
};
struct SoeYxNew
{
    MLang::Long       chan_id;
    MLang::Long       keyid;
    char      value;
    int       status;
    MLang::Long       second;
    short     msecond;
```

```
    int         reserved1;
    int         reserved2;
    void __write(MLang::OutputStream& __os)const;
    void __read(MLang::InputStream& __is);
};
typedef MLang::VECTOR<SoeYxNew> SoeYx_SEQ_New;
struct SoeYxPkgNew_XP
{
    FesPkgNewHead   package_head;
    SoeYx_SEQ_New soe_seq;
    SoeYxPkgNew_XP();
    SoeYxPkgNew_XP(const SoeYxPkgNew_XP&);
    SoeYxPkgNew_XP& operator=(const SoeYxPkgNew_XP&);
    void __write(MLang::OutputStream& __os)const;
    void __read(MLang::InputStream& __is);
};
struct ChangeYcNew
{
    MLang::Long       chan_id;
    MLang::Long       keyid;
    float     value;
    int         status;
    MLang::Long       second;
    short     msecond;
    int         reserved1;
    int         reserved2;
    void __write(MLang::OutputStream& __os)const;
    void __read(MLang::InputStream& __is);
};
typedef MLang::VECTOR<ChangeYcNew> ChangeYc_SEQ_New;
struct ChangeYcPkgNew_XP
{
    FesPkgNewHead   package_head;
    ChangeYc_SEQ_New            yc_seq;
    ChangeYcPkgNew_XP();
    ChangeYcPkgNew_XP(const ChangeYcPkgNew_XP&);
    ChangeYcPkgNew_XP& operator=(const ChangeYcPkgNew_XP&);
```

```
    void __write(MLang::OutputStream& __os)const;
    void __read(MLang::InputStream& __is);
};
struct CtrlResultNew
{
    MLang::Long        chan_id;
    MLang::Long        keyid;
    int         result;
    int         index_no;
    MLang::Long        second;
    short      msecond;
    int         reserved1;
    int         reserved2;
    void __write(MLang::OutputStream& __os)const;
    void __read(MLang::InputStream& __is);
};
typedef CtrlResultNew CtrlResult_SEQ_New;
struct CtrlResultPkgNew_XP
{
    FesPkgNewHead   package_head;
    CtrlResult_SEQ_New            result;
    void __write(MLang::OutputStream& __os)const;
    void __read(MLang::InputStream& __is);
};
```

A.22.3 应用向前置下发验收信息文件定义

A.22.3.1 文件命名：厂站 ID_通道 ID_simu_test.cime，文件格式见表 A.278：

表 A.278 应用向前置下发验收信息文件定义

文件内容	名称	参数说明	是否必填
type	测点类型	表示 yx：遥信、yc：遥测、dc：遥控 pd：调档 sp：设点等	*
num	测点个数	验收测点个数	*
dot_no	点号	测点点号	
keyid	测点 ID	测点 keyid	*
desc	中文描述	测点中文描述	*
reference	对象引用	测点 reference	*
fc	功能约束	功能约束	*
obj_value	目标值	验收目标值	*
cur_value	当前值	测点当前值	*

A.22.3.2 文件示例：

<!Entity=自动验收__ time='2020-11-25_15:48:17' Version=1.0 Code=UTF-8!>
<验收类型>
@sn type num
//序号　测点类型　测点个数
#1　yx　2
</验收类型>
<Object::测点>
@sn　dot_no　key_id　desc reference fc　obj_value　cur_value
//序号　点号　测点 ID　中文描述　对象引用　功能约束　目标值　当前值
#1　1　122160226379078063　时钟同步装置北斗信号状态 SCV0002XLD0/LCSM1.HostRef1Alarm.stVal ST　1　0
#2　2　122160226379078149　时钟同步装置过程层被监测设备 05 测量服务告警 SCV0002XLD1/G1TMSM5.HostMeasAlarm.stVal ST 1 0
</Object::测点>

A.23　CASE断面管理

A.23.1 从实时库保存当前断面数据

A.23.1.1 断面管理说明

建立断面管理程序时必须构造一个 JKIModeManage 类的对象，由该类的接口负责对 CASE 断面进行操作。该类主要提供接口函数完成 CASE 断面数据的保存，获取和删除等操作。

包含头文件：jk_api/jk_IModeManage.h。

连接动态库：libjk_case_mode_manage_client.so。

A.23.1.2 函数原型：int SaveModeFromRTDB(const struct InputModeInfoParaStruct& t_inputmodeinfopara, int& mode_id)

A.23.1.3 函数说明：从实时库保存当前断面数据。

A.23.1.4 参数说明见表 A.279：

表 A.279 从实时库保存当前断面数据

接口参数	输入/输出	参数（返回值）说明	备注
struct InputModeInfoParaStruct& t_inputmodeinfopara	In	要保存的断面信息	—
int& mode_id	Out	保存成功的断面 ID	—
返回值	Out	>0：成功 <0：失败	—

A.23.1.5 示例程序

```
JKIModeManage* mode_op;
int occur_time = (int)time(NULL);
int mode_id;
InputModeInfoParaStruct t_inputmodeinfopara;
t_inputmodeinfopara.mode_time = occur_time;
strcpy(t_inputmodeinfopara.mode_name,  "test_mode");
strcpy(t_inputmodeinfopara.user_name,  "user");
strcpy(t_inputmodeinfopara.mode_desc, "tes_mode_sample");
t_inputmodeinfopara.app_no = AP_SCADA;
int ret = mode_op.SaveModeFromRTDB(t_inputmodeinfopara,  mode_id);
```

A.23.2 查询某一时间段内某一应用族内所有的断面信息

A.23.2.1 函数原型：int ModeQueryByPeriodOfTime(time_t start_current_time,time_t end_current_time,int app_fam, vector<mode_info_struct>& vec_get_mode_info)

A.23.2.2 函数说明：查询某一时间段内某一应用族内所有的断面信息。

A.23.2.3 参数说明见表 A.280：

表 A.280 查询某一时间段内某一应用族内所有的断面信息

接口参数	输入/输出	参数（返回值）说明	备注
time_t start_current_time	In	查询起始时间	—
time_t end_current_time	In	查询结束时间	—
int app_fam	In	应用族号	—
vector<mode_info_struct>& vec_get_mode_info	Out	满足条件的所有断面信息	—
返回值	Out	>0：成功 <0：失败	—

A.23.2.4 示例程序

```
JKIModeManage* mode_op;
time_t end_current_time = (int)time(NULL);
time_t start_current_time = end_current_time - 86400;
int app_fam = AF_SCADA;
vector<mode_info_struct> vec_get_mode_info;
int ret = mode_op.ModeQueryByPeriodOfTime( start_current_time, end_current_time, app_fam, vec_get_mode_info);
```

A.23.3 下装CASE断面数据到实时库

应满足以下要求：

a） 函数原型：int ModeDownLoadToRTDB(const int mode_id, const int app_fam, short ctx_no, int app_no)

b） 函数说明：下装 CASE 断面数据到实时库。

c） 参数说明见表 A.281：

表 A.281 下装 CASE 断面数据到实时库

接口参数	输入/输出	参数（返回值）说明	备注
const int mode_id	In	历史 CASE 断面 ID	—
int app_fam	In	应用族号	—
short ctx_no	In	断面下载到实时库的态号	—
int app_no	In	断面下载到实时库的应用号	—
返回值	Out	>0：成功 <0：失败	—

A.23.4 根据断面ID删除断面数据

应满足以下要求：

a） 函数原型：int ModeDeleteById(const int mode_id, const int app_no)。

b） 函数说明：根据断面 ID 删除断面数据。

c） 参数说明见表 A.282：

表 A.282 根据断面 ID 删除断面数据

接口参数	输入/输出	参数（返回值）说明	备注
const int mode_id	In	历史 CASE 断面 ID	—
const int	In	断面所属应用	—
返回值	Out	>0：成功 <0：失败	—

A.24 CASE模型操作

A.24.1 CASE 模型说明

建立模型操作程序时必须构造一个 JKICaseModelOp 类的对象，由该类的接口负责对 CASE 模型进行操作。该类主要提供接口函数完成 CASE 模型数据的保存，获取和删除等操作。头文件及动态库名称参见基础说明 2.8.1。

A.24.2 保存CASE模型数据

A.24.2.1 函数原型：int ModelSave(int& case_id, const TCaseTemplateInfo& m_t_mstempinfo, const char* case_desc, const char* user_name)。

A.24.2.2 函数说明：保存 CASE 模型数据。

A.24.2.3 参数说明见表 A.283：

表 A.283 保存 CASE 模型数据

接口参数	输入/输出	参数（返回值）说明	备注
int& case_id,	Out	保存成功的模型 CaseID	—
TCaseModelSave& t_casemodelsave	In	需要保存的 CASE 模型信息	—
const char* case_desc	In	保存的 CASE 模型描述信息	—
const char* user_name	In	保存的 CASE 模型的用户名	—

A.24.2.4 示例程序：

```
JKICaseModelOp* model_op;
int occur_time = (int)time(NULL);
TCaseTemplateInfo temp_case_info;
strcpy(temp_case_dlginfo.case_name, "test1");
temp_case_dlg.case_contexttype = AC_REALTIME_NO;
temp_case_dlg.case_type       = CASEMODELTYPE ;

temp_case_dlg.vec_appno.push_back(AP_SCADA);
temp_case_dlg.vec_appno.push_back(AP_PUBLIC);
temp_case_dlginfo.case_appsum = temp_case_dlginfo.vec_appno.size();
temp_case_dlginfo.app_id = 3;
int case_id;
char case_desc[64];
strcpy(case_desc, "test_model");
char user_name[32];
strcpy(user_name, "d5000");
model_op.ModelSave(case_id, temp_case_dlginfo, case_desc, user_name);
```

A.24.3 根据时间点获取CASE最匹配的模型

A.24.3.1 函数原型：int CaseQueryByTime(time_t current_time, int app_no, TCaseInfo& t_case_info,int query_policy = 0)。

A.24.3.2 函数说明：保存 CASE 模型数据。

A.24.3.3 参数说明见表 A.284：

表 A.284 根据时间点获取 CASE 最匹配的模型

接口参数	输入/输出	参数（返回值）说明	备注
time_t current_time	In	需要匹配的 CASE 模型时间点	—
int app_no	In	需要匹配的 CASE 模型应用号	—
int query_policy	In	CASE 模型匹配的策略	0：根据时间

			点往后匹配 1：根据时间 点往前匹配
const char* user_name	In	保存的 CASE 模型的用户名	—
返回值	Out	>0：成功 <0：失败	—

A.24.3.4 示例程序：

```
int occur_time = occur_time - 86400;
int app_no = AP_SCADA;
TCaseInfo t_case_info;
int ret = model_op.CaseQueryByTime(occur_time, app_no, t_case_info, 0);
```

A.24.4 CASE模型下载到实时库中

A.24.4.1 函数原型：int CaseDownLoadToLocal(const short cur_ctx_no, const int ca—seid, const int case_appno, int download_app_no)。

A.24.4.2 函数说明：把 CASE 模型下载到实时库中。

A.24.4.3 参数说明见表 A.285：

表 A.285 CASE 模型下载到实时库中

接口参数	输入/输出	参数（返回值）说明	备注
const short cur_ctx_no	In	CASE 模型需要下载到的态号	s
const int caseid	In	CASE 模型 ID	—
int case_appno	In	CASE 模型的应用号	—
int download_app_no	In	CASE 模型需要下载到的应用号	—
返回值	Out	>0：成功 <0：失败	—

A.24.4.4 示例程序：

```
int occur_time = occur_time - 86400;
int app_no = AP_SCADA;
TCaseInfo t_case_info;
int ret = model_op.CaseQueryByTime(occur_time, app_no, t_case_info, 0);
if (ret > 0)
{
        ret = model_op.CaseDownLoadToLocal(2, t_case_info.case_id, AP_SCADA, AP_SCADA);
}
```

A.24.5 删除CASE模型

应满足以下要求：

a） 函数原型：int CaseDeleteById(const int case_id)。

b） 函数说明：删除 CASE 模型。

c） 参数说明见表 A.286：

表 A.286 删除 CASE 模型

接口参数	输入/输出	参数（返回值）说明	备注
const int case_id	In	CASE 模型 ID	—
返回值	Out	>0：成功 <0：失败	—

A.25 数据采集转发变电站网络安全事件接口

数据采集通过消息总线的方式转发从变电站采集的网络安全事件：

a） 消息通道：JK_API_MSGCHAN_S_FES_NETSEC。

b） 消息类型：JK_API_MSGTYPE_S_NETSEC_EVENT、JK_API_MSGTYPE_S_NETSEC_LINK

c） 网安事件信息根据变电站上送情况实时触发发送给网安应用，链路信息可周期发送给应用。

应满足以下消息结构要求：

```
struct PkgHead
{
   int package_type;
   int data_num;
   int para1;
   int para2;
   MLang::Long para3;
   void __write(MLang::OutputStream& __os)const;
   void __read(MLang::InputStream& __is);
};
struct Data_Buf
{
   MLang::Long dcd_id;
   char        data_buf[2048];

   char        dev_name[64];
   char        ipstr[4];
   int data_len;
   MLang::Long recv_msec;
   MLang::Long post_msec;
   char        reason[1024];
   void __write(MLang::OutputStream& __os)const;
   void __read(MLang::InputStream& __is);
};
typedef MLang::VECTOR<Data_Buf> NetSecData_Buf;
struct Link_Data
{
   MLang::Long dcd_id;
   char        status;
   int flow_bps;
   int flow_pps;
   MLang::Long recv_msec;
   MLang::Long last_online_time;
```

```
    MLang::Long last_offline_time;
    void __write(MLang::OutputStream& __os)const;
    void __read(MLang::InputStream& __is);
};
typedef MLang::VECTOR<Link_Data> NetSecData_Link;
struct NetSecPkg_Data
{
    PkgHead       package_head;
    NetSecData_Buf          netsec_data;
    NetSecPkg_Data();
    NetSecPkg_Data(const NetSecPkg_Data&);
    NetSecPkg_Data& operator=(const NetSecPkg_Data&);
    void __write(MLang::OutputStream& __os)const;
    void __read(MLang::InputStream& __is);
};
struct NetSecPkg_Link
{
    PkgHead       package_head;
    NetSecData_Link          netsec_link;
    NetSecPkg_Link();
    NetSecPkg_Link(const NetSecPkg_Link&);
    NetSecPkg_Link& operator=(const NetSecPkg_Link&);
    void __write(MLang::OutputStream& __os)const;
    void __read(MLang::InputStream& __is);
};
```

A.26 消息通道及事件类型接口

根据消息通道字符串宏获取消息通道值应满足以下要求：

a） 函数原型：int GetChannelNoByString(const std::string sChannelKey)

b） 函数说明：根据消息通道字符串宏获取消息通道值。

c） 参数说明见表 A.287：

表 A.287 根据消息通道字符串宏获取消息通道值

接口参数	输入/输出	参数（返回值）说明	备注
const std::string sChannelKey	Int	消息通道字符串宏定义名称	—
返回值	Out	>0：成功，返回消息通道值 <0：失败	—

根据消息事件字符串宏获取消息事件值应满足以下要求：

a） 函数原型： int GetEventNoByString(const std::string sEventKey)。
b） 函数说明：根据消息事件字符串宏获取消息事件值。
c） 参数说明见表 A.288：

表 A.288 根据消息事件字符串宏获取消息事件值

接口参数	输入/输出	参数（返回值）说明	备注
const std::string sEventKey	Int	消息事件字符串宏定义名称	—
返回值	Out	>0：成功，返回消息事件值 <0：失败	—

A.27 跨区文件传输

跨区协同为系统内部 I 区与 IV 区的运行监控与业务提供跨区协同的通用处理，文件双向同步功能包括：

a）I 区向 IV 区可跨正向隔离发送指定文件功能；
b）IV 向 I 区可跨反向隔离发送符合标准格式的 E 文件：
1） 包含头文件： jk_trans_filesendclient.h；
2） 连接动态库：libjk_trans_filesendclient.so；
3） 函数原型：int FileTransSend(char *file_name, char *dest_name, vector<string> dest_dirs)；
4） 函数说明：跨区发送本地文件接口；
5） 参数说明见表 A.289：

表 A.289 跨区发送本地文件接口

接口参数	输入/输出	参数（返回值）说明	备注
char *file_name,	In	本地文件名称	包含文件路径
char *dest_name	In	对应的转发配置名称~~或远端域名~~	名称为英文字符串，使用者自定义对应的配置文件名称，并在对应的代理服务器上配置转发配置文件~~或远端域名~~
vector<string> dest_dirs	In	接收端文件保存的相对路径名称	保存路径名称不包含文件名称，支持保存在多个目录下
返回值	Out	1：发送成功 <0：发送失败	—

6） 示例程序：

```
#include "jk_trans_filesendclient.h"
int main(int argc, char *argv[])
{

    string src_path_and_file = "/home/ecs/send.txt";
    string dest_name = "file_srv";
    vector<string> dest_dirs;
    dest_dirs.push_back("var/trans_test");
    dest_dirs.push_back("var/trans_test1");

    // 构造发送类
```

```
    CJKFileSendClient transSend;
    // 发送文件
    int ret_code = transSend.FileTransSend(src_path_and_file.c_str(), dest_name.c_str(), dest_dirs);
    cout << "transSend.FileTransSend() return " << ret_code << endl;

        if (1 == ret_code){
            cout << "send file success "  << endl;
        return 0;
        }
        else{
            cout << "send file error"  << endl;
        return -1;
        }
}
```

A.28 服务总线与安全认证整合接口

该章节接口将 A.10.3.2 服务类业务认证加密接口与 A.9 服务总线接口中的安全认证、服务请求加密与服务请求进行功能合并，简化应用调用方式。

A.28.1 安全认证初始化接口

应满足以下要求：

安全认证初始化：

1) 接口原型:：int JKSecServiceAuthInit (char* srv_name, char* firm_name, char* pwd)
2) 接口说明：该接口用于应用和后台服务的安全认证的初始化。
3) 参数列表见表 A.290：

表 A.290 JKSecServiceAuthInit 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
srv_name	char*	In	服务名，全英文	是
firm_name	char*	In	厂商名，全英文	是
pwd	char*	In	P12文件的保护口令	是
返回值	int	Out	=0：成功; <0：失败	—

A.28.2 服务端安全接口

应满足以下要求：

a) 服务端初始化：
   1) 接口原型：int JkSecServiceServerInit(JK_ServiceInfo serviceinfo, int mode)
   2) 接口说明：该接口用于请求/响应模型服务端的初始化。
   3) 参数列表见表 A.291：

表 A.291 JkSecServiceServerInit 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
serviceinfo	JK_ServiceInfo	In	服务的连接信息（如地址、端口号等）	是
mode	int	In	服务端运行方式，默认（mode=JK_DISPATCH）	是
返回值	int	Out	=1：成功; <0：失败	—

b) 服务分发-单次响应方式：
   1) 接口原型：

typedef void* (*JK_Func)(char* requestBuffer, int requestlen, char** responseBuffer, int* responselen);
int JkSecServiceDispatch (JK_ServiceInfo serviceinfo, int flag, JK_Func func)

2) 接口说明：该接口完成请求报文的接收和任务的分发，有多线程和单线程两种方式。对于多线程方式，接口为每个请求启动一个线程，接受请求报文，然后调用 func 回调函数，要求 func 必须是线程安全的，函数内部对于关键区域必须互斥。对于单线程方式，不再启动线程。

3) 参数列表见表 A.292：

表 A.292 JkSecServiceDispatch 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
serviceinfo	JK_ServiceInfo	In	服务的连接信息（如地址、端口号等）	是
flag	int	In	服务端运行方式（多线程：2，单线程：0）	是
func	JK_Func	In	回调函数	是
requestBuffer	char*	In	要处理的请求报文	是
requestlen	int	In	请求报文长度	是
responseBuffer	char**	Out	响应报文	是
responselen	int*	Out	响应报文长度	是
返回值	int	Out	=1：成功; <0：失败	—
注意：回调函数 func 中的 responseBuffer 指向的响应报文所需内存由回调函数使用 malloc 分配，服务总线调用回调函数后将释放该内存。				

c) 订阅内容注册函数：

1) 接口原型：typedef int JK_Determine(const char *requestBuffer, const int requestlen);
int JkSecServicePublishRegister(JK_ServiceInfo serviceinfo, JK_Determine* determine_func = NULL, const int handle_num = 1)

2) 接口说明：注册订阅/发布结果类型。

3) 参数列表见表 A.293：

表 A.293 JkSecServicePublishRegister 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
serviceinfo	JK_ServiceInfo	In	服务的连接信息（如地址、端口号等）	是
determine_func	Determine*	In	订阅内容与发布句柄分发函数。 返回值为发布句柄	是
handle_num	int	In	发布句柄个数，根据服务端发布的主题个数来定	是
返回值	int	Out	=0：成功; <0：失败	—

d) 订阅结果发布：

1) 接口原型：int JkSecServicePublish(const char *responseBuffer, const int responselen, const int handle_index = 0)

2) 接口说明：订阅结果发布函数。

3) 参数列表见表 A.294：

表 A.294 JkSecServicePublish 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
responseBuffer	char**	In	发布结果报文	是
responselen	int*	In	发布结果报文长度	是
handle_index	int	In	发布结果句柄，为 JkServicePublishRegister 中注册的 JK_Determine 函数中返回的句柄， 必须小于	是

			handle_num，如果只提供一种类型的结果发布，该参数不填写	
返回值	int	Out	=0：成功 =-1：失败 =-2：无订阅	—

A.28.3 局域客户端安全接口

a） 同步请求服务函数：

1) 接口原型：int JkSecServiceRequestSync (JK_ServiceInfo serviceinfo, const char *requestBuffer, int requestlen, time_t timeout, char ** responseBuff, int *responselen, JK_Handle *requesthandle)

2) 接口说明：客户端提交服务请求，阻塞等待结果返回。

3) 参数列表见表 A.295：

表 A.295 JkSecServiceRequestSync 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
serviceinfo	JK_ServiceInfo	In	服务连接信息， 由 JkServiceLocate 返回	是
requestBuffer	const char *	In	请求报文	是
requestlen	int	In	请求报文长度	是
timeout	time_t	In	超时时间	是
responseBuff	char **	Out	响应报文	—
responselen	int *	Out	响应报文长度	—
requesthandle	JK_Handle *	Out	请求句柄，用于 JkServiceRequestFree 和 JkServiceHandleFree 及链路复用。由原语返回	—
返回值	int	Out	=0：成功; <0：失败	—
注：第一次使用时 requesthandle 初始化为 JK_HANDLE_INIT，以后直接使用即可，复用同一条链路；在返回出错后，建议释放请求句柄，重新初始化 JK_HANDLE_INIT 为进行重新连接（以下链路复用的接口与此类似）。				

b） 异步请求接口：

1) 接口原型：int JkSecServiceRequestAsync (JK_ServiceInfo serviceinfo, const char *requestBuffer, int requestlen, time_t timeout, JK_Handle *requesthandle)

2) 接口说明：客户端提交基于 S 语言的服务请求，结果由 JkServiceReqSyncTest（）函数返回。不需要返回结果则调用服务请求资源释放函数即可。

3) 参数列表见表 A.296：

表 A.296 JkSecServiceRequestAsync 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
serviceinfo	JK_ServiceInfo	In	服务连接信息， 由服务管理返回	是
requestBuffer	const char *	In	请求报文	是

requestlen	int	In	请求报文长度	是
timeout	time_t	In	超时时间	是
requesthandle	JK_Handle *	Out	请求句柄，用于 JkServiceRequestFree 和 JkServiceHandleFree 及链路复用。由原语返回	—
返回值	int	Out	=0：成功 <0：失败	—

c） 异步请求结果：

1) 接口原型：int JkSecServiceReqSyncTest (JK_Handle requesthandle, char ** responseBuff, int buflen)

2) 接口说明：读取异步请求的结果，如果请求任务未完成，则阻塞等待结果返回；如果已完成，但是失败，则返回失败。

3) 参数列表见表 A.297：

表 A.297 JkSecServiceReqSyncTest 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
requesthandle	JK_Handle	In	请求句柄，由 JkServiceRequestAsync 返回	是
responseBuff	char **	Out	响应报文	—
responselen	int *	Out	响应报文长度	—
返回值	int	Out	<0：失败 =1：成功	—
注释：每次调用异步请求结果接口后均需调用 JkServiceRequestFree 资源释放接口释放请求服务函数申请的资源。				

d） 服务请求资源释放函数：

1) 接口原型：int JkSecServiceRequestFree (JK_Handle requesthandle)

2) 接口说明：释放请求服务函数申请的资源。

3) 参数列表见表 A.298：

表 A.298 JkSecServiceRequestFree 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
requesthandle	JK_Handle	In	请求句柄	是
返回值	int	Out	=0：成功; <0：失败	—

e） 句柄释放函数：

1) 接口原型：int JkSecServiceHandleFree (JK_Handle requesthandle)

2) 接口说明：释放请求句柄。

3) 参数列表见表 A.299：

表 A.299 JkSecServiceHandleFree 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填

requesthandle	JK_Handle	In	请求句柄	是
返回值	int	Out	=0：成功; <0：失败	—

f） 服务订阅函数：

1) 接口原型：Typedef int JK_Function (char * responseBuffer, int buflen);int JkSecServiceSubscribe(JK_ServiceInfo serviceinfo, int flag, const char *requestBuffer, int requestlen, JK_Function* func, JK_Handle *handle)

2) 接口说明：该接口完成服务的订阅。首先按照 flag 制定的方式订阅服务，然后等待订阅确认，如果确认失败，则直接结束；如果确认成功，则创建线程，等待订阅消息的返回，每收到一个回送来的结果，就调用回调函数 func 执行数据处理。

3) 参数列表见表 A.300：

表 A.300 JkSecServiceSubscribe 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
serviceinfo	JK_ServiceInfo	In	服务连接信息， 由 JkServiceLocate 返回	是
requestBuffer	const char *	In	请求报文	是
requestlen	int	In	请求报文长度	是
flag	int	In	同步，默认为 0	否
handle	JK_Handle *	Out	请求句柄，用于 JkServiceRequestFree 和 JkServiceHandleFree 及链路复用。由原语返回	—
responseBuffer	char *	Out	响应报文	—
buflen	int	Out	响应报文长度	—
返回值	int	Out	=0：成功; <0：失败	—
注：订阅接口每个 handle 只支持订阅一个主题。				

g） 订阅取消函数：

1) 接口原型：int JkSecServiceUnSubscribe(JK_Handle handle)

2) 接口说明：该接口取消对应的内容。

3) 参数列表见表 A.301：

表 A.301 JkSecServiceUnSubscribe 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
handle	JK_Handle	In	请求句柄	是
返回值	int	Out	=0：成功; <0：失败	—

A.28.4 广域客户端安全接口

应满足以下要求：

a) 远程同步请求接口：

1) 接口原型：

```
int JkSecRemoteRequestSync(JK_DomainInfo        *domaininfo,
                           const char           *requestBuffer,
                           int                  requestlen,
                           time_t               timeout,
                           char                 **responseBuff,
                           int                  *responselen,
                           JK_Handle            *handle,
                           int                  resendFlag = 1,
                           const char           *IP = NULL)
```

2) 接口说明：该接口用于远程服务请求（单次请求单次响应），阻塞等待结果返回，可以设置等待超时时间。

3) 参数列表见表 A.302：

表 A.302 JkSecRemoteRequestSync 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
domaininfo	JK_DomainInfo *	In	域信息	是
requestBuffer	char *	In	请求报文	是
requestlen	int	In	请求报文长度	是
timeout	time_t	In	超时时间，默认 60 秒	否
responseBuff	char **	Out	响应报文	—
responselen	int *	Out	响应报文长度	—
handle	JK_Handle *	Out	请求句柄	—
resendFlag	int	In	重传标志	否
IP	const char *	In	用于连接指定 IP 地址代理	否
返回值	int	Out	>0：成功 <=0 失败	—
注： 3) 第一次使用时 handle 初始化为 JK_HANDLE_INIT，以后直接使用即可，复用同一条链路； 4) 接口调用失败不需要释放任何资源（接口内部已自动释放）。				

b) 远程请求服务资源释放接口：

1) 接口原型：int JkSecRemoteRequestFree(JK_Handle requesthandle)

2) 接口说明：释放请求服务函数申请的资源，不调用此接口会导致内存泄漏。

3) 参数列表见表 A.303：

表 A.303 JkSecRemoteRequestSync 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
requesthandle	JK_Handle	In	请求句柄	是
返回值	int	Out	=0：成功; <0：失败	—

c) 远程请求句柄释放接口：

1) 接口原型：int JkSecRemoteHandleFree (JK_Handle requesthandle)

2) 接口说明：释放请求句柄，关闭连接。

3) 参数列表见表 A.304：

表 A.304 JkSecRemoteRequestSync 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
requesthandle	JK_Handle	In	请求句柄	是
返回值	int	Out	=0：成功; <0：失败	—

d) 广域订阅请求接口:

1) 接口原型：typedef int JK_Function (char *responseBuffer, int buflen);
int JkSecRemoteSubscribe(JK_DomainInfo *domaininfo,const char *requestBuffer,int requestlen,JK_Function *func,JK_Handle *handle,int flag = 0 ,const char *IP = NULL)；

2) 接口说明：该接口完成广域服务的订阅。首先按照 flag 制定的方式订阅服务，然后等待订阅确认，如果确认失败，则直接结束；如果确认成功，等待订阅消息的返回，每收到一个回送来的结果，就调用回调函数 func 执行数据处理。

3) 参数列表见表 A.305：

表 A.305 JkSecRemoteSubscribe 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
domaininfo	JK_DomainInfo *	In	域信息	是
requestBuffer	Const char *	In	请求报文	是
requestlen	int	In	请求报文长度	是
func	JK_Function*	In	回调函数指针，用于接收服务端推送信息。	是
flag	Int	In	同步，默认为 0	否
handle	JK_Handle *	Out	请求句柄，用于 JkRemoteUnSubscribe，由该接口返回。	—
responseBuffer	char *	Out	响应报文	—
buflen	int	Out	响应报文长度	—

IP	const char *	In	用于连接指定 IP 地址代理，默认为 NULL，一般不需要填写。	否
返回值	Int	Out	>0：成功 <=0 失败	—

e) 广域服务订阅取消接口:

1) 接口原型：int JkSecRemoteUnSubscribe (JK_Handle requesthandle)；

2) 接口说明：取消订阅内容。

3) 参数列表见表 A.306：

表 A.306 JkSecRremoteUnSubscribe 接口参数列表

接口参数	类型	输入/输出	参数说明	是否必填
requesthandle	JK_Handle	In	请求句柄	是
返回值	int	Out	=0：成功; <0：失败	—

A.28.5 局域服务安全访问调用示例

a) 局域请求响应模型调用示例如下:

1) 服务端

```
#include ”jk_sec_services.h”
JK_Func Do(char* requestBuffer, int requestlen, char** responseBuffer, int* responselen)
{
    * responselen = xxxx;
    *responseBuffer = (char *)malloc(* responselen );
        // 填写应答内容
    return (void *)1;

}
int main(int argc, char **argv)
{
    int port = xxxx;// 服务端口号
    JK_ServiceInfo serviceinfo;
    serviceinfo.port = port;
    int servid = xxx;//服务 ID，建议与服务端口保持一致
    int balance = 0;
                    char* firm_name = “kedong”;//厂商名
                char* pwd = “1234”;//服务的保护口令
                int ret = JKSecServiceAuthInit(serviceinfo.servname,firm_name, pwd);
                if(ret !=0)
                  return 0;//认证失败，自动退出.....
    JkSecServiceServerInit(serviceinfo, JK_DISPATCH);
    JkServiceRegisterInit(“dispatch”, “realtime”, “public”, servid ,  port, balance);
  JkSecServiceDispatch (serviceinfo, 2, Do);
  while(1) { // 防止主进程退出
        sleep(10);
      }
}
```

2) 客户端

```
#include  "jk_sec_services.h"
int main(int argc, char **argv)
```

```
{
    JK_ServiceInfo serviceinfo;
    char* firm_name = “kedong”;//厂商名
    char* pwd = “1234”;//服务的保护口令
    int ret = JKSecServiceAuthInit(serviceinfo.servname,firm_name, pwd);
    if(ret !=0)
                    return 0;//认证失败，自动退出
    //定位服务所在的节点
    int ret = JkServiceLocate(ctx_name, app_name, proc_name, &serviceinfo,balance);
    serviceinfo.addr = htonl(serviceinfo.addr);
    JK_Handle handle = JK_HANDLE_INIT;
    char *resp_buffer;
    int respe_len;
    //同步请求
    ret = JkSecServiceRequestSync(serviceinfo, request, request_len, 10, &resp_buffer,
    &resp_len, &handle);
    if(0 > ret)  {
        printf("serviceRequestSync error : %s\n",  ret);
    }  else {
        printf("response buff len %d\n", resp_len);
        printf("response buff:%s\n", resp_buffer);
    }
    JkSecServiceRequestFree(handle);   //释放请求服务函数申请的资源
    JkSecServiceHandleFree(handle);   //释放请求句柄
    return 0;
}
```

b) 订阅发布模型调用示例如下:

1) 服务端

```
JK_Determine determine_func(const char *requestBuffer, const int requestlen)
                                                    // //*requestBuffer 要处理的请求报文
{
    if (0 == strncmp(requestBuffer, "SE", strlen("SE")))
        return 1;
    else if (0 == strncmp(requestBuffer, "DSA", strlen("DSA")))
        return 2;
     return -1;
}
int main(int argc, char **argv)
{
    JK_ServiceInfo serviceinfo;
    serviceinfo.port = port;
    char  pbuf[1024];
    int buf_len =1024,loop = 0;
    char* firm_name = “kedong”;//厂商名
    char* pwd = “1234”;//服务的保护口令
    int ret = JKSecServiceAuthInit(serviceinfo.servname,firm_name, pwd);
    if(ret !=0)
      return 0;//认证失败，自动退出
    ret = JkSecServicePublishRegister(serviceinfo,determine_func ,2);
    if (0 > ret) {
```

```
            printf("JkServicePublishRegister error\n"); exit(1);
        }
        while (1){
            JkSecServicePublish(pbuf, buf_len, 1)); //订阅结果发布，1 为 determine_func 函数中返回的句柄
            sleep(1);
        }
    }
```

2) 客户端

```
    JK_Function scribeHandle(const char * responseBuffer, int buflen)// * responseBuffer 响应报文
    {
        // 对应答内容进行显示或者处理
    }
    int main(int argc, char **argv)
    {
        JK_ServiceInfo serviceinfo;
        char* firm_name = “kedong”;//厂商名
        char* pwd = “1234”;//服务的保护口令
        int ret = JKSecServiceAuthInit(serviceinfo.servname,firm_name, pwd);
        if(ret !=0)
                return 0;//认证失败，自动退出
        //定位服务所在的节点
        int ret = JkServiceLocate(ctx_name, app_name, proc_name, &serviceinfo);
        serviceinfo.addr = htonl(serviceinfo.addr);
        JK_Handle handle = JK_HANDLE_INIT;
        ret = JkSecServiceSubscribe(serviceinfo, isAsync, request,req_len, scribeHandle, &handle);
        if(0 > ret) {
            printf(stderr, " serviceSubscribe error : %s\n", ret);
        }
        else {
            printf(“serviceSubscribe ok!\n”);
        }
        sleep(10);
        JkSecServiceUnSubscribe(handle);
        return 0;
    }
```

A.28.6 广域服务安全访问调用示例

a) 广域请求响应模型客户端调用示例如下:

```
#include "jk_sec_remoteCallClient.h"

int main(int argc, char **argv)
{
    if (argc != 5) {
        printf("用法:%s [态号] [应用号] [服务名] [域名]\n", argv[0]);
        printf("%d\n",argc);
        return -1;
    }

    JK_DomainInfo domaininfo;
```

```
        domaininfo.context = atoi(argv[1]);
        domaininfo.appno = atoi(argv[2]);
        strcpy(domaininfo.servicename, argv[3]);
        strcpy(domaininfo.domain, argv[4]);

        time_t timeout = 10;
        char *responseBuff = NULL;
        int responselen = 0;
        JK_Handle handle = JK_HANDLE_INIT;

        char requestBuffer[1024] = "hello";

        int requestlen = strlen(requestBuffer);

        char* firm_name = “kedong”;//厂商名

        char* pwd = “1234”;//服务的保护口令
        int ret = JKSecServiceAuthInit(domaininfo.servicename,firm_name, pwd);
        if(ret !=0)
                return 0;//认证失败，自动退出

        ret = JkSecRemoteRequestSync(&domaininfo, requestBuffer, requestlen, timeout,
&responseBuff, &responselen, &handle);

        if (0 >= ret) {
            printf("服务总线同步请求原语调用失败!\n");
            return -1;
        }

        printf("+++++ 远程同步请求接口调用成功，ret = %d +++++\n", ret);
        printf("响应报文长度:%d\n", responselen);
        printf("响应报文:%s\n", responseBuff);

        JkSecRemoteHandleFree(handle);
        JkSecRemoteUnSubscribe(handle);

        return 0;
            }
```

b) 广域订阅发布模型客户端调用示例如下:

```
JK_Function scribeHandle(const char * responseBuffer, int buflen)
{
  // 对响应内容进行显示或者处理
}
int main(int argc, char **argv)
{
    if (argc != 5) {
            printf("用法:%s [态号] [应用号] [服务名] [域名]\n", argv[0]);
            printf("%d\n",argc);
            return -1;
        }

        JK_DomainInfo domaininfo;
        domaininfo.context = atoi(argv[1]);
        domaininfo.appno = atoi(argv[2]);
        strcpy(domaininfo.servicename, argv[3]);
```

```
        strcpy(domaininfo.domain, argv[4]);
    JK_Handle handle = JK_HANDLE_INIT;  //请求句柄
    char requestBuffer[1024] = "xxx";//请求报文
    int requestlen = strlen(requestBuffer);//请求报文长度

        char* firm_name = “kedong”;//厂商名

        char* pwd = “1234”;//服务的保护口令
        int ret = JKSecServiceAuthInit(domaininfo.servicename,firm_name, pwd);
        if(ret !=0)
            return 0;//认证失败，自动退出
        ret = JkSecRemoteSubscribe(&domaininfo, requestBuffer, requestlen, scribeHandle,
&handle, 0);
    //该接口完成广域服务的订阅。首先按照 flag 制定的方式订阅服务，然后等待订阅确认，如
    果确认失败，则直接结束；如果确认成功，等待订阅消息的返回，每收到一个回送来的结果，
    就调用回调函数 func 执行数据处理。
        if (0 >= ret) {
                printf("服务总线同步请求原语调用失败!\n");
                return -1;
        }
            sleep(100);
      JkSecRemoteUnSubscribe(handle);// 取消订阅内容。
      printf("success:%d\n", nSucceed);
      return 0;}
```

A.29 安全Ⅳ区JAVA消息总线接口

A.29.1 消息注册

A.29.1.1 接口原型：public static void registerListener(EvtRecvListener listener,String channel)，接口包名为 jk.service. evt，接口类名为 IEvtservService；

A.29.1.2 接口说明：使用消息总线的所有进程都需要先初始化。初始化主要完成对文件映射等初始化工作。

A.29.1.3 参数列表见表 A.307：

表 A.307 messageInit 原语参数列表

接口参数	类型	输入输出	参数说明	是否必填
listener	EvtRecvListener	In	监听对象	是
channel	String	In	通道号	是

A.29.1.4 调用示例

```
public class xxxBusinessEvtListener implements EvtRecvListener {
    public void initEventListener() {
        IEvtservService.registerListener(this,BUSINESS_CHANNEL); // channelKBUSINESS_CHANNEL:通道
宏定义
    }
}
```

A.29.2 取消订阅

A.29.2.1 接口原型：public static void removeEvent(EvtRecvListener listener) ，接口包名为 jk.service. evt，接口类名为 IEvtservService。

A.29.2.2 接口说明：订阅某一通道的消息后，因需要取消订阅该通道的消息。将本进程 id 从通道进

程组中抹去。

A.29.2.3 参数列表见表 A.308：

表 A.308 messageUnSubscribe 原语参数列表

接口参数	类型	输入输出	参数说明	是否必填
listener	EvtRecvListener	In	监听对象	是

A.29.2.4 调用示例

```
public class xxxBusinessEvtListener implements EvtRecvListener {
    public void initEventListener() {
        IEvtservService.removeEvent(this); // 先删除监听再注册监听
        IEvtservService.registerListener(this,BUSINESS_CHANNEL); // BUSINESS_CHANNEL:通道宏
定义
    }
}
```

A.29.3 消息接收

A.29.3.1 接口原型：public void evtRecvPerformed(String eventid, String msg)，接口包名为 jk.service.evt，接口类名为 EvtRecvListener。

A.29.3.2 接口说明：调用该接口可以接收已注册通道的消息。

A.29.3.3 参数列表见表 A.309：

表 A.309 messageReceive 原语参数列表

接口参数	类型	输入输出	参数说明	是否必填
eventid	String	In	接收消息事件号	是
msg	String	In	接收消息内容	否

A.29.3.4 调用示例

```
public class xxxBusinessEvtListener implements EvtRecvListener {
    public void initEventListener() {
        IEvtservService.removeEvent(this); // 先删除监听再注册监听
        IEvtservService.registerListener(this,BUSINESS_CHANNEL); //BUSINESS_CHANNEL:通道宏
定义
    }
    @Override
    public void evtRecvPerformed(String eventid, String msg){
            // 事件号过滤
                if (eventid != Constants.BUSINESS_EVENT_NO) {
                    return;
                }
            // 解析 msg 的 json 结构，获取业务数据
            JSONObject json = JSONObject.parseObject(result);
            ..... // 业务处理
        }
    }
```

A.29.4 消息发送

A.29.4.1 接口原型：public static boolean sendEvent(String channel,String eventName, String msg)，接口包名为 jk.service. evt，接口类名为 IEvtservService。

A.29.4.2 接口说明：调用该接口可以发送指定通道的消息到消息总线。
A.29.4.3 参数列表见表 A.310：

表 A.310 messageSend 原语参数列表

接口参数	类型	输入输出	参数说明	是否必填
channel	String	In	通道号	是
eventid	String	In	事件号	是
msg	String	In	发送消息体	是
返回值	boolean	Out	true：成功; false：失败。	是

A.29.4.4 调用示例

```
String msg=  "{"key1":intValue,"key2":"strValue"}" ;
// 向指定通道发送指定事件号的消息
boolean isSendOk = IEvtservService.sendEvent(BUSINESS_CHANNEL, BUSINESS_EVENT_NO,msg);
If（!isSendOk）{
 // 业务异常处理
}
```

A.30 消息通道分配说明

附录 A26 消息通道及事件类型接口中主要消息通道宏定义有：

表 A.311 消息通道宏定义

消息通道宏定义	说明
JK_API_MSGCHAN_S_WARN_INFORM	告警订阅通道
JK_API_MSGCHAN_S_TICKET_SEND	顺控操作调用消息通道客户端->服务端
JK_API_MSGCHAN_S_TICKET_RECV	顺控操作调用消息通道服务端->客户端
JK_API_MSGCHAN_S_AUTOACCEPTANCE	自动对点消息通道遥信验收
JK_API_MSGCHAN_S_AUTOACCEPTANCE_CTRL	自动对点消息通道遥控验收
JK_API_MSGCHAN_S_FES_NETSEC	网络安全事件消息通道
JK_API_MSGCHAN_S_MAIN_REAL_DATA	主设备实时数据订阅通道
JK_API_MSGCHAN_S_AUX_REAL_DATA	辅设备实时数据订阅通道
JK_API_MSGCHAN_S_MAIN_LIGHT_CONFIGM	主设备光字牌确认
JK_API_MSGCHAN_S_AUX_LIGHT_CONFIGM	辅设备光字牌确认
JK_API_MSGCHAN_S_VIDEOLINKAGE	视频联动消息通道

JK_API_MSGCHAN_S_VIDEOCONFIRM	视频确认结果消息通道

附 录 B
（资料性）
安全区Ⅰ、Ⅱ服务化接口定义

B.1 基本原则

基础平台应为第三方应用的开发、运行和管理提供通用的技术支撑。集控系统基于平台提供系统管理服务、历史数据服务、实时数据服务、文件服务、日志服务、告警服务、权限服务等多种服务功能，提供统一的基础平台公共数据访问接口。第三方应用通过公共服务访问接口获取平台提供的数据，并将处理的结果返回给基础平台。基础平台以服务接口+数据信息结构的方式提供完整的信息访问机制。第三方应用与基础平台之间的数据交互方式包括主动查询、周期通知、触发通知、权限校验等，要求如下：

a) 主动查询：应用通过主动调用服务接口的方式请求查询需要的数据，应满足以下要求：
   1) 主动查询的数据支持查询条件过滤；
   2) 查询的数据类型包括实时数据、历史数据、历史告警数据和文件数据等。
b) 周期通知：基础平台周期向应用发送订阅的数据，通知周期由平台根据服务类型确定，应满足以下要求：
   1) 应用向基础平台进行数据订阅，订阅后基础平台周期向应用发送订阅的实时数据；
   2) 订阅的周期通知数据支持查询条件过滤。
c) 触发通知：基础平台在数据变化时向应用发送订阅的发布数据，应满足以下要求：
   1) 应用向基础平台进行数据订阅，订阅后基础平台实时数据变化时向应用主动发送订阅的发布数据；
   2) 订阅的触发通知数据支持查询条件过滤。
d) 权限校验，应满足以下要求：
   1) 应用调用平台接口请求进行权限校验；
   2) 应用通过基础平台进行应用自身界面的用户登录校验；
   3) 应用通过基础平台进行应用自身界面的操作权限校验；
   4) 应用向平台请求用户登录或权限校验时，由平台接口或平台服务端弹出统一的用户登录和权限校验窗口并返回校验结果，统一的校验窗口应可进行用户名、密码、双因子和相应请求权限的综合验证；
   5) 用户登录校验成功后基础平台返回校验成功的结果、校验通过的用户名及具备的该应用操作权限码列表；
   6) 用户权限校验成功后基础平台返回校验成功的结果及校验通过的用户名。

基础平台向第三方应用提供安全认证接口，第三方应用在请求操作控制服务前，应调用基础平台安全认证接口进行安全认证。安全认证校验通过后，基础平台接收到第三方应用的操作控制服务请求时，采用通信加密技术对交互数据进行加密控制，以保证操作控制类信息传递的机密性、完整性。

B.2 数据信息结构定义

B.2.1 总体要求

数据信息结构是由基础平台预先定义的、可由第三方应用直接访问的数据结构。数据信息结构与基础平台内部数据结构之间的映射由基础平台自己实现。

B.2.2 基础数据信息结构

基础数据信息结构采用 XML 文件描述，保存为名称为 platform_base.xml 的数据信息结构文件中，具体内容参见表 B.1。

数据信息结构内容包括 platform_base.xml 中定义的数据结构，在此基础上，基础平台和第三方应用可根据业务需要进行信息结构内容扩展。基础数据信息结构见表 B.1。

表 B.1 基础数据信息结构

```
<?xml version="1.0" encoding="utf-8"?>
<platform>
    <!--数据信息结构定义-->
    <datatypes>
        <datatype name="subcontrolarea" desc="区域" type="data">
            <field name="id" desc="标识" type="ft_int64" />
            <field name="name" desc="中文名称" type="ft_string"/>
            <field name="father_id" desc="父区域ID" type="ft_int64"/>
        </datatype>
        <datatype name="substation" desc="厂站" type="data">
            <field name="id" desc="标识" type="ft_int64" />
            <field name="name" desc="中文名称" type="ft_string"/>
            <field name="st_type" desc="厂站类型" type="ft_int"/>
            <field name="subarea_id" desc="所属区域标识" type="ft_int64"/>
            <field name="bv_id" desc="最高电压类型id" type="ft_int64"/>
        </datatype>
        <datatype name="basevoltage" desc="电压类型" type="data">
            <field name="id" desc="标识" type="ft_int64" />
            <field name="name" desc="中文名称" type="ft_string"/>
            <field name="nomvol" desc="基准电压" type="ft_double"/>
        </datatype>
        <datatype name="bay" desc="间隔" type="data">
            <field name="id" desc="标识" type="ft_int64"/>
            <field name="name" desc="中文名称" type="ft_string"/>
            <field name="st_id" desc="所属厂站标识" type="ft_int64"/>
            <field name="bv_id" desc="所属电压等级标识" type="ft_int64"/>
        </datatype>
        <datatype name="powertransformer" desc="变压器" type="data">
            <field name="id" desc="标识" type="ft_int64"/>
            <field name="name" desc="中文名称" type="ft_string"/>
            <field name="tr_type" desc="类型" type="ft_int"/>
            <field name="st_id" desc="厂站标识" type="ft_int64"/>
            <field name="bay_id" desc="所属间隔" type="ft_int64"/>
            <field name="tpcolor" desc="拓扑着色" type="ft_int"/>
        </datatype>
        <datatype name="transformerwinding" desc="变压器绕组" type="data">
            <field name="id" desc="标识" type="ft_int64"/>
            <field name="name" desc="中文名称" type="ft_string"/>
            <field name="wind_type" desc="绕组类型" type="ft_int"/>
            <field name="st_id" desc="厂站标识" type="ft_int64"/>
            <field name="bay_id" desc="所属间隔" type="ft_int64"/>
            <field name="tr_id" desc="变压器标识" type="ft_int64"/>
            <field name="nde" desc="物理连接节点" type="ft_int64"/>
            <field name="bv_id" desc="基准电压标识" type="ft_int64"/>
            <field name="p" desc="有功值" type="ft_double"/>
            <field name="p_qual" desc="有功值质量码" type="ft_int"/>
            <field name="q" desc="无功值" type="ft_double"/>
```

```
        <field name="q_qual" desc="无功值质量码" type="ft_int"/>
        <field name="i" desc="电流值" type="ft_double"/>
        <field name="i_qual" desc="电流值质量码" type="ft_int"/>
        <field name="tap" desc="分接头位置" type="ft_double"/>
        <field name="tap_qual" desc="分接头质量码" type="ft_int"/>
        <field name="tpcolor" desc="拓扑着色" type="ft_int"/>
        <field name="knd" desc="中性点连接点号" type="ft_int64"/>
    </datatype>
    <datatype name="breaker" desc="断路器" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="name" desc="中文名称" type="ft_string"/>
        <field name="brk_type" desc="断路器类型" type="ft_int"/>
        <field name="ind" desc="首端连接节点号" type="ft_int64"/>
        <field name="jnd" desc="末端连接节点号" type="ft_int64"/>
        <field name="st_id" desc="所属厂站标识" type="ft_int64"/>
        <field name="bay_id" desc="所属间隔" type="ft_int64"/>
        <field name="bv_id" desc="基准电压标识" type="ft_int64"/>
        <field name="point" desc="遥信值" type="ft_int"/>
        <field name="status" desc="状态" type="ft_int"/>
        <field name="tpcolor" desc="拓扑着色" type="ft_int"/>
    </datatype>
    <datatype name="disconnector" desc="刀闸" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="name" desc="中文名称" type="ft_string"/>
        <field name="ind" desc="首端连接节点" type="ft_int64"/>
        <field name="jnd" desc="末端连接节点" type="ft_int64"/>
        <field name="st_id" desc="所属厂站标识" type="ft_int64"/>
        <field name="bay_id" desc="所属间隔" type="ft_int64"/>
        <field name="bv_id" desc="基准电压标识" type="ft_int64"/>
        <field name="point" desc="遥信值" type="ft_int"/>
        <field name="status" desc="状态" type="ft_int"/>
        <field name="tpcolor" desc="拓扑着色" type="ft_int"/>
    </datatype>
    <datatype name="grounddisconnector" desc="接地刀闸" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="name" desc="中文名称" type="ft_string"/>
        <field name="st_id" desc="所属厂站标识" type="ft_int64"/>
        <field name="bay_id" desc="所属间隔" type="ft_int64"/>
        <field name="bv_id" desc="基准电压标识" type="ft_int64"/>
        <field name="nd" desc="连接点号" type="ft_int64"/>
        <field name="point" desc="遥信值" type="ft_int"/>
        <field name="status" desc="状态" type="ft_int"/>
        <field name="tpcolor" desc="拓扑着色" type="ft_int"/>
    </datatype>
    <datatype name="compensator_p" desc="并联补偿器" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="name" desc="中文名称" type="ft_string"/>
        <field name="st_id" desc="所属厂站标识" type="ft_int64"/>
        <field name="bay_id" desc="所属间隔" type="ft_int64"/>
        <field name="bv_id" desc="基准电压标识" type="ft_int64"/>
```

```
        <field name="nd" desc="连接点号" type="ft_int64"/>
        <field name="cp_type" desc="补偿器类型" type="ft_int"/>
        <field name="q" desc="无功值" type="ft_double"/>
        <field name="q_qual" desc="无功值质量码" type="ft_int"/>
        <field name="i_a_value" desc="A相电流值" type="ft_double"/>
        <field name="i_a_qual" desc="A相电流值质量码" type="ft_int"/>
        <field name="i_b_value" desc="B相电流值" type="ft_double"/>
        <field name="i_b_qual" desc="B相电流值质量码" type="ft_int"/>
        <field name="i_c_value" desc="C相电流值" type="ft_double"/>
        <field name="i_c_qual" desc="C相电流值质量码" type="ft_int"/>
        <field name="tpcolor" desc="拓扑着色" type="ft_int"/>
    </datatype>
    <datatype name="compensator_s" desc="串联补偿器" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="name" desc="中文名称" type="ft_string"/>
        <field name="st_id" desc="所属厂站标识" type="ft_int64"/>
        <field name="bay_id" desc="所属间隔" type="ft_int64"/>
        <field name="bv_id" desc="基准电压标识" type="ft_int64"/>
        <field name="ind" desc="首端连接节点号" type="ft_int64"/>
        <field name="jnd" desc="末端连接节点号" type="ft_int64"/>
        <field name="tpcolor" desc="拓扑着色" type="ft_int"/>
    </datatype>
    <datatype name="energyconsumer" desc="负荷" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="name" desc="名称" type="ft_string"/>
        <field name="st_id" desc="厂站标识" type="ft_int64"/>
        <field name="bay_id" desc="所属间隔" type="ft_int64"/>
        <field name="nd" desc="物理连接节点" type="ft_int64"/>
        <field name="bv_id" desc="基准电压标识" type="ft_int64"/>
        <field name="p" desc="有功值" type="ft_double"/>
        <field name="p_qual" desc="有功值质量码" type="ft_int"/>
        <field name="q" desc="无功值" type="ft_double"/>
        <field name="q_qual" desc="无功值质量码" type="ft_int"/>
        <field name="i" desc="电流值" type="ft_double"/>
        <field name="i_qual" desc="电流值质量码" type="ft_int"/>
        <field name="tpcolor" desc="拓扑着色" type="ft_int"/>
    </datatype>
    <datatype name="acline" desc="线路" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="name" desc="中文名称" type="ft_string"/>
        <field name="bv_id" desc="基准电压标识" type="ft_int64"/>
    </datatype>
    <datatype name="aclinesegment" desc="交流线段" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="name" desc="中文名称" type="ft_string"/>
        <field name="ist_id" desc="一端变电站ID" type="ft_int64"/>
        <field name="jst_id" desc="二端变电站ID" type="ft_int64"/>
        <field name="bv_id" desc="基准电压标识" type="ft_int64"/>
        <field name="ind" desc="首端连接节点号" type="ft_int64"/>
```

```
        <field name="jnd" desc="末端连接节点号" type="ft_int64"/>
        <field name="tpcolor" desc="拓扑着色" type="ft_int"/>
    </datatype>
    <datatype name="aclineend" desc="线段端点" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="name" desc="中文名称" type="ft_string"/>
        <field name="st_id" desc="变电站ID" type="ft_int64"/>
        <field name="bay_id" desc="间隔ID" type="ft_int64"/>
        <field name="aclnseg_id" desc="线路ID" type="ft_int64"/>
        <field name="bv_id" desc="电压类型ID" type="ft_int64"/>
        <field name="nd" desc="连接点号" type="ft_int64"/>
        <field name="p" desc="有功值" type="ft_double"/>
        <field name="p_qual" desc="有功质量码" type="ft_int"/>
        <field name="q" desc="无功值" type="ft_double"/>
        <field name="q_qual" desc="无功质量码" type="ft_int"/>
        <field name="i" desc="电流值" type="ft_double"/>
        <field name="i_qual" desc="电流质量码" type="ft_int"/>
        <field name="tpcolor" desc="拓扑着色" type="ft_int"/>
    </datatype>
    <datatype name="busbarsection" desc="母线" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="name" desc="中文名称" type="ft_string"/>
        <field name="st_id" desc="变电站ID" type="ft_int64"/>
        <field name="bay_id" desc="间隔ID" type="ft_int64"/>
        <field name="bv_id" desc="电压类型ID" type="ft_int64"/>
        <field name="nd" desc="连接点号" type="ft_int64"/>
        <field name="v" desc="线电压" type="ft_double"/>
        <field name="v_qual" desc="线电压质量码" type="ft_int"/>
        <field name="ang" desc="相角" type="ft_double"/>
        <field name="ang_qual" desc="相角质量码" type="ft_int"/>
        <field name="f" desc="频率" type="ft_double"/>
        <field name="f_qual" desc="频率质量码" type="ft_int"/>
        <field name="tpcolor" desc="拓扑着色" type="ft_int"/>
    </datatype>
    <datatype name="mutualinductor" desc="互感器" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="name" desc="中文名称" type="ft_string"/>
        <field name="st_id" desc="变电站ID" type="ft_int64"/>
        <field name="bay_id" desc="间隔ID" type="ft_int64"/>
        <field name="bv_id" desc="电压类型ID" type="ft_int64"/>
        <field name="nd" desc="连接点号" type="ft_int64"/>
        <field name="inductor_type" desc="互感器类型" type="ft_int"/>
        <field name="tpcolor" desc="拓扑着色" type="ft_int"/>
    </datatype>
    <datatype name="arcsuppressioncoil" desc="消弧线圈" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="name" desc="中文名称" type="ft_string"/>
        <field name="st_id" desc="变电站ID" type="ft_int64"/>
        <field name="bay_id" desc="间隔ID" type="ft_int64"/>
        <field name="bv_id" desc="电压类型ID" type="ft_int64"/>
```

```
        <field name="nd" desc="连接点号" type="ft_int64"/>
        <field name="tpcolor" desc="拓扑着色" type="ft_int"/>
    </datatype>
    <datatype name="lightningarrester" desc="避雷器" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="name" desc="中文名称" type="ft_string"/>
        <field name="st_id" desc="变电站ID" type="ft_int64"/>
        <field name="bay_id" desc="间隔ID" type="ft_int64"/>
        <field name="bv_id" desc="电压类型ID" type="ft_int64"/>
        <field name="nd" desc="连接点号" type="ft_int64"/>
        <field name="tpcolor" desc="拓扑着色" type="ft_int"/>
    </datatype>
    <datatype name="cntrl_ied" desc="二次基本信息" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="name" desc="中文名称" type="ft_string"/>
        <field name="ied_name" desc="iedName" type="ft_string"/>
        <field name="st_id" desc="变电站ID" type="ft_int64"/>
        <field name="bay_id" desc="间隔ID" type="ft_int64"/>
        <field name="vl_id" desc="电压类型ID" type="ft_int64"/>
        <field name="type" desc="类型" type="ft_int"/>
        <field name="model" desc="型号" type="ft_string"/>
        <field name="addr" desc="103通讯地址" type="ft_int"/>
        <field name="soft_version" desc="软件版本" type="ft_string"/>
    </datatype>
    <datatype name="cntrl_ied_ldevice" desc="二次定值区号" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="name" desc="中文名称" type="ft_string"/>
        <field name="st_id" desc="变电站ID" type="ft_int64"/>
        <field name="ied_id" desc="二次设备ID" type="ft_int64"/>
        <field name="bay_id" desc="间隔ID" type="ft_int64"/>
        <field name="min_sg" desc="最小定值区号" type="ft_int"/>
        <field name="max_sg" desc="最大定值区号" type="ft_int"/>
        <field name="act_sg" desc="当前定值区号" type="ft_double"/>
        <field name="act_sg_qual" desc="当前定值区号质量码" type="ft_int"/>
    </datatype>
    <datatype name="auxiliary_device" desc="辅助设备" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="dev_name" desc="中文名称" type="ft_string"/>
        <field name="st_id" desc="变电站ID" type="ft_int64"/>
        <field name="aux_class" desc="设备类别" type="ft_int"/>
        <field name="aux_type" desc="子类型" type="ft_int"/>
    </datatype>
    <datatype name="measanalog" desc="主设备遥测" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="alg_id" desc="遥测ID" type="ft_int64"/>
        <field name="name" desc="中文名称" type="ft_string"/>
        <field name="st_id" desc="变电站ID" type="ft_int64"/>
        <field name="upt_time" desc="更新时间" type="ft_string"/>
        <field name="chg_time" desc="变化时间" type="ft_string"/>
```

```
        <field name="reference" desc="reference属性" type="ft_string"/>
    </datatype>
    <datatype name="measpoint" desc="主设备遥信" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="pnt_id" desc="遥信ID" type="ft_int64"/>
        <field name="name" desc="中文名称" type="ft_string"/>
        <field name="st_id" desc="变电站ID" type="ft_int64"/>
        <field name="upt_time" desc="更新时间" type="ft_string"/>
        <field name="chg_time" desc="变化时间" type="ft_string" />
        <field name="reference" desc="reference属性" type="ft_string"/>
    </datatype>
    <datatype name="auxiliary_yc" desc="辅助设备遥测" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="yc_name" desc="中文名称" type="ft_string"/>
        <field name="st_id" desc="变电站ID" type="ft_int64"/>
        <field name="dev_id" desc="所属设备ID" type="ft_int64"/>
        <field name="yc_value" desc="遥测值" type="ft_double"/>
        <field name="qual" desc="质量码" type="ft_int"/>
        <field name="upt_time" desc="更新时间" type="ft_string"/>
        <field name="chg_time" desc="变化时间" type="ft_string"/>
        <field name="reference" desc="参引" type="ft_string"/>
        <field name="yc_type" desc="遥测类别" type="ft_int"/>
        <field name="yc_sub_type" desc="遥测类型" type="ft_int"/>
    </datatype>
    <datatype name="auxiliary_yx" desc="辅助设备遥信" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="yx_name" desc="中文名称" type="ft_string"/>
        <field name="st_id" desc="变电站ID" type="ft_int64"/>
        <field name="dev_id" desc="所属设备ID" type="ft_int64"/>
        <field name="yx_value" desc="遥信值" type="ft_int"/>
        <field name="qual" desc="遥信质量码" type="ft_int"/>
        <field name="upt_time" desc="更新时间" type="ft_string"/>
        <field name="chg_time" desc="变化时间" type="ft_string"/>
        <field name="reference" desc="参引" type="ft_string"/>
        <field name="yx_type" desc="遥信类别" type="ft_int"/>
        <field name="yx_sub_type" desc="遥信类型" type="ft_int"/>
    </datatype>
    <datatype name="relaysig" desc="保护信号" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="name" desc="中文名称" type="ft_string"/>
        <field name="st_id" desc="变电站ID" type="ft_int64"/>
        <field name="bay_id" desc="间隔ID" type="ft_int64"/>
        <field name="value" desc="值" type="ft_int"/>
        <field name="qual" desc="质量码" type="ft_int"/>
    </datatype>
    <datatype name="electriccub" desc="网门" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="name" desc="中文名称" type="ft_string"/>
        <field name="st_id" desc="变电站ID" type="ft_int64"/>
        <field name="bay_id" desc="间隔ID" type="ft_int64"/>
```

```
        <field name="bv_id" desc="电压类型ID" type="ft_int64"/>
        <field name="value" desc="值" type="ft_int"/>
        <field name="qual" desc="质量码" type="ft_int"/>
    </datatype>
    <datatype name="groundstake" desc="接地桩线" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="name" desc="中文名称" type="ft_string"/>
        <field name="st_id" desc="变电站ID" type="ft_int64"/>
        <field name="bay_id" desc="间隔ID" type="ft_int64"/>
        <field name="bv_id" desc="电压类型ID" type="ft_int64"/>
        <field name="tpcolor" desc="拓扑着色" type="ft_int"/>
        <field name="nd" desc="连接点号" type="ft_int64"/>
        <field name="point" desc="遥信值" type="ft_int"/>
        <field name="qual" desc="质量码" type="ft_int"/>
    </datatype>
    <datatype name="substation_area" desc="变电站区域" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="area_name" desc="中文名称" type="ft_string"/>
        <field name="st_id" desc="变电站ID" type="ft_int64"/>
    </datatype>
    <datatype name="substation_cabinet" desc="变电站屏柜" type="data">
        <field name="id" desc="标识" type="ft_int64"/>
        <field name="name" desc="中文名称" type="ft_string"/>
        <field name="st_id" desc="变电站ID" type="ft_int64"/>
        <field name="bay_id" desc="间隔ID" type="ft_int64"/>
        <field name="substation_area" desc="变电站区域ID" type="ft_int64"/>
    </datatype>
    <datatype name="sys_table_info" desc="表信息" type="data">
        <field name="table_id" desc="表号" type="ft_int"/>
        <field name="table_name_eng" desc="表英文名称" type="ft_string"/>
        <field name=" table_name_chn" desc="表中文名称" type=" ft_string "/>
    </datatype>
    <datatype name="sys_column_info" desc="域信息" type="data">
        <field name="table_id" desc="表号" type="ft_int"/>
        <field name="field_id" desc="域号" type="ft_int"/>
        <field name="column_name_eng" desc="域英文名称" type="ft_string"/>
        <field name=" column_name_chn" desc="域中文名称" type=" ft_string "/>
        <field name="data_type" desc="数据类型" type="ft_int"/>
        <field name="data_length" desc="数据长度" type="ft_int"/>
        <field name="menu_name" desc="菜单名" type="ft_string"/>
    </datatype>
    <datatype name="sys_menu_info" desc="菜单信息" type="data">
        <field name="menu_name" desc="菜单名" type="ft_string"/>
        <field name="menu_no" desc="序号" type="ft_int"/>
        <field name="actual_value" desc="实际值" type=" ft_int"/>
        <field name="display_value" desc="显示值" type=" ft_string"/>
    </datatype>
    <datatype name="token_define" desc="标志牌定义" type="data">
        <field name="token_no" desc="标识" type="ft_int64"/>
        <field name="name" desc="中文名称" type="ft_string"/>
```

```
            <field name="token_type" desc="标志牌类型" type=" ft_int"/>
            <field name="icon_name" desc="图元名称" type=" ft_string"/>
        </datatype>
        <datatype name="typedef" desc="类型定义" type="data">
            <field name="id" desc="标识" type="ft_int64"/>
            <field name="description" desc="描述" type="ft_string"/>
            <field name="tablename" desc="表名称" type="ft_string"/>
            <field name="fieldname" desc="字段名称" type="ft_string"/>
            <field name="typevalue" desc="各表中类型字段的值列表" type="ft_string"/>
            <field name="typedesc" desc="各表中类型字段的描述列表" type="ft_string"/>
        </datatype>
    </datatypes>
    <!--类型树定义-->
    <stationtrees>
        <!--一次系统信息结构-->
        <equiptree name="equiptypetree" desc="一次系统信息结构">
            <typeinfo name="subcontrolarea" key="id" primarykey="">
                <typeinfo name="substation" key="id" primarykey="subarea_id">
                    <typeinfo name="bay" key="id" primarykey="st_id">
                        <typeinfo name="powertransformer" key="id"
primarykey="bay_id">
                            <typeinfo name="transformerwinding" key="id"
primarykey="tr_id"/>
                        </typeinfo>
                        <typeinfo name="busbarSection" key="id" primarykey="bay_id"/>
                        <typeinfo name="breaker" key="id" primarykey="bay_id"/>
                        <typeinfo name="disconnector" key="id" primarykey="bay_id"/>
                        <typeinfo name="grounddisconnector" key="id"
primarykey="bay_id"/>
                        <typeinfo name="compensator_p" key="id" primarykey="bay_id"/>
                        <typeinfo name="compensator_s" key="id" primarykey="bay_id"/>
                        <typeinfo name="energyconsumer" key="id"
primarykey="bay_id"/>
                        <typeinfo name="aclineend" key="id" primarykey="bay_id"/>
                    </typeinfo>
                    <typeinfo name="measanalog" key="id" primarykey="st_id"/>
                    <typeinfo name="measpoint" key="id" primarykey="st_id"/>
                </typeinfo>
            </typeinfo>
        </equiptree>
        <!--辅助系统信息结构-->
        <equiptree name="auxiliarytypetree" desc="辅助系统信息结构">
            <typeinfo name="substation" key="id" primarykey="">
                <typeinfo name="auxiliary_device" key="id" primarykey="st_id">
                    <typeinfo name="auxiliary_yc" key="id" primarykey="dev_id"/>
                    <typeinfo name="auxiliary_yx" key="id" primarykey="dev_id"/>
                </typeinfo>
            </typeinfo>
        </equiptree>
        <!--二次采集信息结构-->
        <iedtree name="iedtypetree" desc="二次采集信息结构">
            <typeinfo name="substation" key="id" primarykey="">
                <typeinfo name="bay" key="id" primarykey="st_id">
                    <typeinfo name="cntrl_ied" key="id" primarykey="bay_id">
```

```
                                <typeinfo name="cntrl_ied_ldevice" key="id"
primarykey="ied_id"/>
                            </typeinfo>
                        </typeinfo>
                    </typeinfo>
                </iedtree>
            </stationtrees>
</platform>
```

注1：typedef数据信息结构为平台提供的其它业务数据结构中类型字段的定义信息，通过接口可获取信息结构中所有属于类型字段的所有定义的数值与描述信息，数据信息结构中的时间数据使用yyyy-MM-dd HH:mm:ss.SSS时间格式。

注2：第三方应用可根据业务需要在B2.2基础数据信息结构的基础上进行信息结构内容扩展，进行表的自定义设计，基础平台向第三方应用提供自定义表的访问，第三方应用调用平台接口，访问自定义表中的数据信息。

B.2.3 告警数据信息结构

B.2.3.1 告警信息结构采用 XML 文件描述，保存为名称为 platform_alarm.xml 的数据信息结构文件中，具体内容参见表 B.2。

B.2.3.2 告警信息结构内容包括 platform_alarm.xml 中定义的数据结构，在此基础上，基础平台和第三方应用可根据业务需要进行信息结构内容扩展。告警数据信息结构见表 B.2。

表 B.2 告警数据信息结构

```
<?xml version="1.0" encoding="utf-8"?>
<platform>
    <!--模型定义-->
    <datatypes>
        <datatype name="alarm" desc="告警" type="data">
            <field name="objid" desc="告警对象ID" type="ft_int64"/>
            <field name="objname" desc="告警对象名称" type="ft_string"/>
        <field name="alarmtype" desc="告警类型" type="ft_int"/>
            <field name="alarmlevel" desc="告警等级" type="ft_int" />
            <field name="alarmgroup" desc="告警组" type="ft_string/>
        <field name="alarmitem" desc="告警项" type="ft_string/>
            <field name="content" desc="告警内容" type="ft_string"/>
            <field name="subid" desc="厂站ID" type="ft_int64"/>
            <field name="subname" desc="厂站名称" type="ft_string"/>
        <field name="bayid" desc="间隔ID" type="ft_int64"/>
            <field name="bayname" desc="间隔名称" type="ft_string"/>
        <field name="equipmentid" desc="一次设备ID" type="ft_int64"/>
            <field name="equipmentname" desc="一次设备名称" type="ft_string"/>
            <field name="alarmtime" desc="告警时间" type="ft_string"/>
        <field name="aornum" desc="责任区号" type="ft_int64"/>
        <field name="status" desc="状态" type="ft_int"/>
            <field name="ackuser" desc="确认人" type="ft_string"/>
            <field name="acktime" desc="确认时间" type="ft_string"/>
            </datatype>
    </datatypes>
</platform>
```

注：时间数据使用yyyy-MM-dd HH:mm:ss.SSS时间格式，alarmgroup为告警大类，比如“越限监视”，alarmitem为告警大类下的子项，比如“越上限” 。当告警类型为遥信告警或SOE告警时，status表示对应的位置信息，1表示分位，2表示合位。

B.2.3.3 告警等级参考表 B.3，告警类型参考表 B.4 。

表 B.3 告警级别定义

告警级别	严重程度	常量码
事故	紧急	1
异常	重要	2
越限	次要	3
变位	一般	4
告知	一般	5

表 B.4 告警类型定义

告警类型	常量码
系统告警	1
操作记录	2
维护日志	3
遥测告警	4
遥信告警	5
遥控告警	6
SOE告警	7
通讯状态	8

B.2.4 历史采样信息结构

B.2.4.1 历史采样信息结构采用 XML 文件描述，保存为名称为 platform_hisdata.xml 的数据信息结构文件中，具体内容参见表 B.5。

B.2.4.2 历史采样信息结构内容包括 platform_hisdata.xml 中定义的数据结构，在此基础上，基础平台和第三方应用可根据业务需要进行信息结构内容扩展。 历史采样信息结构见表 B.5。

表 B.5 历史采样信息结构

```
<?xml version="1.0" encoding="utf-8"?>
<platform>
    <!--模型定义-->
    <datatypes>
        <datatype name="hisdata" desc="历史采样信息" type="data">
            <field name="objid" desc="对象ID" type="ft_int64"/>
            <field name="objtype" desc="对象类型" type="ft_string"/>
            <field name="datetime" desc="保存时间" type="ft_string"/>
            <field name="value" desc="值" type="ft_double"/>
        </datatype>
     <datatype name="hissta" desc="历史采样统计信息" type="data">
            <field name="objid" desc="对象ID" type="ft_int64"/>
            <field name=”statimeunit” desc=”统计时间单位” type=”ft_string”/>
       <field name="stadatetime" desc="统计时间段起点" type="ft_string"/>
            <field name="maxvalue" desc="最大值" type="ft_double"/>
       <field name="maxvaluetime" desc="最大值对应时间" type="ft_string"/>
```

```
        <field name="minvalue" desc="最小值" type="ft_double"/>
        <field name="minvaluetime" desc="最小值对应时间" type="ft_string"/>
        <field name="avgvalue" desc="平均值" type="ft_double"/>
         </datatype>
    </datatypes>
</platform>
```

注：时间数据使用yyyy-MM-dd HH:mm:ss.SSS时间格式。

B.2.5 审计数据信息结构

B.2.5.1 审计数据信息结构采用 XML 文件描述，保存为名称为 platform_auditdata.xml 的数据信息结构文件中，具体内容参见表 B.6。

B.2.5.2 审计数据信息结构内容包括 platform_auditdata.xml 中定义的数据结构，在此基础上，基础平台和第三方应用可根据业务需要进行信息结构内容扩展。审计数据信息结构见表 B.6。

表 B.6 审计数据信息结构

```
<?xml version="1.0" encoding="utf-8"?>
<platform>
    <!--模型定义-->
    <datatypes>
        <datatype name="auditdata" desc="审计数据信息" type="data">
            <field name="username" desc="用户名" type="ft_string"/>
              <field name="datetime" desc="审计时间" type="ft_string"/>
              <field name="auditsubject" desc="审计主体" type="ft_string"/>
              <field name="auditobject" desc="审计客体" type="ft_string"/>
              <field name="audittype" desc="审计类型" type="ft_int"/>
              <field name="auditlevel" desc="审计级别" type=" ft_int"/>
              <field name="auditdesc" desc="审计内容" type="ft_string"/>
        </datatype>
    </datatypes>
</platform>
```

注：用户名为向平台请求的登录用户名或权限校验的用户名，审计主体为审计日志的产生对象，审计客体为审计日志的操作目标对象，时间数据使用yyyy-MM-dd HH:mm:ss.SSS时间格式。

B.2.5.3 审计级别数值定义参考表 B.7，审计类型数值定义参考表 B.8。

表 B.7 审计级别定义

审计级别	常量码
紧急	1
重要	2
次要	3
一般	4

表 B.8 审计类型定义

类型描述	常量码
操作	1
配置	2
查阅	3

B.3 接口形式定义

B.3.1 基本要求

服务接口是由基础平台提供的一组服务函数，中间数据传输使用 UTF-8 编码的 JSON 格式字符串，用于通过数据信息结构读取和修改基础平台的数据，以及收发消息、读写文件等，要求如下：

a) 基础平台提供共享库，为了共享库的通用性，共享库所依赖的第三方 Qt 类库应使用 Qt4.8.4 版本，共享库名称统一命名为 iplatform，公共访问接口由基础平台共享库提供，扩展应用可以动态加载共享库，通过共享库的导出函数 GetPlatform 获取公共访问接口对象，进行数据模型及数据服务的访问；

b) 基础平台提供统一的接口头文件 iplatintf.h，iplatintf.h 文件中的接口及说明见附录 B.3.1、附录 B.3.2 和附录 B.3.3，接口头文件中定义了基础平台与第三方应用交互的标准 C++接口，接口由 IPlatform 访问接口类和 IApplication 回调接口类组成。第三方应用的所有访问请求均使用 IPlatform 访问接口类交互，该接口提供访问请求函数，同时提供回调接口对象注册接口，基础平台返回结果数据使用 IApplication 回调接口类交互。所有接口函数均为纯虚函数（Abstract），IPlatform 接口类由基础平台实现，IApplication 接口类由第三方应用实现，交互内容为 JSON 字符串数据；

c) 平台提供的 SDK 接口程序包由两级目录构成，用于存放平台 SDK 共享数据和应用的应用数据。其中第一级目录为 platformsdk 目录和 application 目录，platformsdk 目录存放平台 sdk 共享数据，application 目录第三方应用的应用数据。platformsdk 目录的第二级目录为 lib 目录、ini 目录、doc 目录和 interface 目录，lib 目录存放平台共享库 platform 文件，ini 目录下存放 platform_base.xml 等信息结构文件，interface 目录存放接口头文件 iplatintf.h，doc 目录存放 sdk 相关的说明文档。

d) application 目录的第二级目录为 bin 目录、lib 目录、doc 目录、ini 目录和 tmp 目录，bin 目录用于存放应用的可执行程序文件，lib 目录用于存放应用的共享动态库文件，ini 目录用于存放第三方应用程序的相关配置文件，tmp 目录用于存放临时保存的文件，doc 目录存放应用相关的说明文档。

B.3.2 导出函数

基础平台共享库提供唯一的对外导出函数 GetPlatform，用于应用程序向基础平台获取平台接口对象，同一进程可以获取不同的 IPlatform 对象，导出函数的具体定义参见表 B.9。

表 B.9 导出函数

```
/*
    【导出函数】    GetPlatform 由基础平台实现，用于应用程序获取平台接口对象指针
*/
bool GetPlatform(IPlatform** ppPlatform);
```

B.3.3 平台接口类

B.3.3.1 平台接口类 IPlatform 由基础平台实现，用于第三方应用向基础平台注册、取消注册以及向基础平台请求数据，平台接口均应保证调用时的线程安全，平台接口类的具体定义参见表 B.10。

表 B.10 平台接口类

```
/*
    【平台接口类】  IPlatform 由基础平台实现，用于向基础平台注册、取消注册以及向基
础平台请求数据
*/
class IPlatform
{
public:
```

```
    /*
        【函数】      纯虚析构函数
    */
    virtual ~ IPlatform() = 0;

    /*
        【函数】      注册应用进程，同步
        【参数】      pApp              应用接口对象
                  sInitJSON         初始化数据
              sOutputJSON     同步返回的结果数据，内容为JSON格式数据
    */
    virtual bool RegisterApp(IApplication* pApp, const std::string& sInitJSON, std::string&
sOutputJSON) = 0;

    /*
        【函数】      注销应用进程
        【参数】      pApp                    应用接口对象
    */
    virtual bool UnregisterApp(IApplication* pApp) = 0;

    /*
        【函数】      同步请求基础平台数据
        【参数】      sInputJSON        输入的请求数据，内容为JSON格式数据
                  sOutputJSON       同步返回的结果数据，内容为JSON格式数据
                  nTimeoutMsec          给定的请求超时等待时间，单位：豪秒
        【返回值】附录B.3.4.1中表B.12定义的交互返回的功能码
    */
    virtual int RequestSyncData(const std::string& sInputJSON, std::string& sOutputJSON, int
nTimeoutMsec) = 0;

    /*
        【函数】      异步请求基础平台数据
        【参数】      sInputJSON            输入请求数据，内容为JSON格式数据
        【返回值】附录B.3.4.1中表B.12定义的交互返回的功能码
    */
    virtual int RequestASyncData(const std::string& sInputJSON) = 0;
};
```

B. 3. 3. 2　注册应用模块的接口函数 RegisterApp：将应用实现的 IApplication 回调接口对象 pApp 注册到平台，注册后用于接收基础平台返回的结果数据，输入参数 sInitJSON 为注册时输入的初始化数据，注册时需要进行的应用权限校验。sOutputJSON 为同步返回的结果数据，内容为 JSON 格式数据。

B. 3. 3. 3　注销应用模块接口函数 UnregisterApp：将应用已注册的回调接口对象 pApp 从基础平台中注销，注销后将无法再收到结果数据。

B. 3. 3. 4　同步请求平台数据接口函数 RequestSyncData：同步请求函数调用接口后阻塞等待结果数据，直到获取到结果数据并赋值 sOutputJSON 后返回。参数 sInputJSON 为已注册的应用模块向平台请求的业务数据，请求内容为 JSON 格式数据；sOutputJSON 为同步返回的结果数据，内容为 JSON 格式数据；nTimeoutMsec 为超时等待时间，超时时间大于 0 时使用该参数的超时时间进行处理，等于或小于 0 时以平台默认值为准；返回值为表 B.12 定义的交互返回的功能码。

B. 3. 3. 5　异步请求平台数据接口函数 RequestASyncData：异步请求函数调用接口后立即返回，不阻塞。参数 sInputJSON 为已注册的应用模块向平台请求的业务数据，请求内容为 JSON 格式数据；返回值为表 B.12 定义的交互返回的功能码。

B.3.4 应用接口类

B.3.4.1 应用接口类 IApplication 由应用实现，用于接收基础平台返回的结果数据，应用接口均应保证调用时的线程安全，应用接口类的具体定义参见表 B.11。

表 B.11 应用接口类

```
/*
    【应用接口类】 IApplication由应用实现，用于接收基础平台返回的结果数据
*/
class IApplication
{
public:
    /*
        【函数】    基础平台异步请求的数据回调接口
        【参数】    sOutputJSON 基础平台返回的请求结果数据，内容为JSON格式数据
    */
    virtual void PltfmResultData(const std::string& sOutputJSON) = 0;

    /*
        【函数】    订阅数据回调接口
        【参数】    sOutputJSON 基础平台发布已订阅的数据，内容为JSON格式数据
    */
    virtual void PltfmIssueData(const std::string& sOutputJSON) = 0;
};
```

B.3.4.2 基础平台数据回调接口 PltfmResultData：参数 sOutputJSON 用于基础平台输出应用异步请求的结果数据，内容为 JSON 格式数据。

B.3.4.3 订阅数据回调接口 PltfmIssueData：参数 sOutputJSON 用于基础平台发布已被订阅的数据，内容为 JSON 格式数据。当周期数据和触发数据同时调用 PltfmIssueData 接口时应用应保证该函数调用的多线程安全。

B.3.5 接口传输数据定义

B.3.5.1 平台应用交互返回的数据码定义

平台应用交互返回的功能码定义参见表 B.12。应用向平台请求校验的权限码定义参见表 B.13，应用的权限应该由平台权限服务统一配置管理，应用的权限校验功能由平台服务接口统一提供。

表 B.12 交互返回的功能码定义

属性	含义	值	含义
result	返回信息功能码	0	成功：返回的单次请求数据
		1	成功：返回的周期变化数据
		2	成功：返回的触发变化数据
		3	失败：注册码校验失败
		4	失败：权限校验失败
		5	失败：请求参数有误
		6	失败：请求结果超时
		7	成功：安全认证通过
		8	失败：安全认证失败
		扩展数值	可根据具体需要约定扩展

表 B.13 应用向平台请求校验的权限码定义

属性	含义	值	含义

right	应用权限码	0	保留
		1	画面浏览权限
		2	模型维护权限
		3	遥控操作权限
		4	告警确认权限
		5	告警清除权限
		6	自动对点权限
		7	信号自动巡视权限
		8	顺控操作权限
		9	顺控监护权限
		扩展数值	可根据具体需要约定扩展

B.3.5.2 平台应用之间的交互数据定义

RequestSyncData 函数输入的是同步请求的参数数据 sInputJSON，输出的是基础平台返回的交互操作结果 sOutputJSON。RequestASyncData 函数输入的是异步请求的参数数据 sInputJSON，PltfmResultData 函数输出的是基础平台异步返回的交互操作结果 sOutputJSON。RequestASyncData 函数同时用于订阅平台周期发送和触发发送的数据，PltfmIssueData 函数输出的是基础平台返回的应用订阅的变化数据 sOutputJSON。返回的交互操作结果由原始请求参数数据和结果数据合并组成。平台与应用交互接口功能定义见表 B.14。

表 B.14 平台与应用交互接口功能定义一览表

功能服务名称	接口功能	接口	dataclass	说明
应用注册	应用注册	RegisterApp		
系统管理	查询应用状态	RequestSyncData（同步） RequestASyncData（异步）	appstate	参见交互数据定义
	查询应用所属主机或备机	RequestSyncData（同步） RequestASyncData（异步）	appnodestate	参见交互数据定义
	查询节点状态	RequestSyncData（同步） RequestASyncData（异步）	nodestate	参见交互数据定义
	查询某一节点所有应用状态	RequestSyncData（同步） RequestASyncData（异步）	nodeappstate	参见交互数据定义
历史数据库	请求历史数据	RequestSyncData（同步） RequestASyncData（异步）	hisdata	参见交互数据定义
	请求历史统计数据	RequestSyncData（同步） RequestASyncData（异步）	Hissta	参见交互数据定义
实时数据库	请求实时数据	RequestSyncData（同步） RequestASyncData（异步）	realdata	参见交互数据定义
文件服务	请求文件数据	RequestSyncData（同步） RequestASyncData（异步）	file	参见交互数据定义
	存储文件数据	RequestSyncData（同步） RequestASyncData（异步）	File	参见交互数据定义
告警服务	请求实时告警	RequestSyncData（同步） RequestASyncData（异步）	realalarm	参见交互数据定义
	应用发送告警	RequestSyncData（同步） RequestASyncData（异步）	realalarm	参见交互数据定义
	请求历史告警	RequestSyncData（同步） RequestASyncData（异步）	hisalarm	参见交互数据定义
权限服务	请求用户登录	RequestSyncData（同步） RequestASyncData（异步）	rightmanage	参见交互数据定义
	请求权限校验	RequestSyncData（同步） RequestASyncData（异步）	rightmanage	参见交互数据定义
	请求权限描述	RequestSyncData（同步） RequestASyncData（异步）	rightmanage	参见交互数据定义

审计服务	存储审计日志	RequestSyncData（同步） RequestASyncData（异步）	auditlog	参见交互数据定义
操作控制服务	安全认证	RequestSyncData（同步）	safetyauth	参见交互数据定义
	控制请求	RequestSyncData（同步） RequestASyncData（异步）	control	参见交互数据定义
	顺控操作调用	RequestSyncData（同步） RequestASyncData（异步）	scontrol	参见交互数据定义
自动对点服务接口	自动对点	RequestSyncData（同步） RequestASyncData（异步）	pointscheck	参见交互数据定义
数据调阅服务接口	文件调阅	RequestSyncData（同步） RequestASyncData（异步）	Defile	参见交互数据定义
	设置动态数据集	RequestSyncData（同步） RequestASyncData（异步）	datasetting	参见交互数据定义
日志服务接口	存储应用日志	RequestSyncData（同步） RequestASyncData（异步）	applog	参见交互数据定义

B.4 应用注册接口

B.4.1 RegisterApp 函数是基础平台提供的应用注册接口，接口中的 sInitJSON 参数为注册应用时需要填写的初始化数据，此接口为输入数据。返回的交互操作结果由原始请求参数数据和结果数据合并组成，以下为交互数据格式定义及 RegisterApp 函数输入和接口同步返回数据的示例，在数据结构定义表中带*的属性项为 sInitJSON 请求数据与 sOutputJSON 返回数据共有的数据结构。

B.4.2 初始化时应用需要读取由人工提供的应用的唯一注册码，注册码为 64 位的加密字符串，由人工使用注册工具计算生成，每个应用程序的注册码应不相同，同一应用程序在不同态中的注册码也应不同，且同一应用程序在不同机器上的注册码也应不同。同一应用程序在同一个态下、同一个机器上只能注册一次。初始化的 JSON 数据参数内容数据格式定义参见表 B.15。

表 B.15 初始化数据的交互数据定义

属性	含义	值	含义
identitycheck	*身份校验	注册码	应用的唯一注册码（64位）
contextname	*应用所属的态名	态名	应用所属的态名
appname	*应用名	应用名	应用的名称
procname	*进程注册名	进程名	进程注册名
result	返回信息功能码	功能码	信息功能码定义参见表B.12

B.4.3 应用向平台注册时的初始化参数数据示例如下：

```
std::string sInitJSON = "{
                        "identitycheck":
                        "WERASDFERSFLKJLKAUYOIUHADKAHKJHUEFJJHASDUHUNZCMVBKJL
                        KSJDFJKHKASN",
                        "contextname": "realtime",
                        "appname": "testapp",
                        "procname": "testproc"
                        }";
```

B.4.4 应用向平台注册返回结果示例如下：

```
std::string sReturnJSON = "{
                          "identitycheck":
                            "WERASDFERSFLKJLKAUYOIUHADKAHKJHUEFJJHASDUHUNZCMV
                            BKJLKSJDFJKHKASN",
                          "contextname": "realtime",
                          "appname": "testapp",
                          "procname": "testproc",
                          "result": 0
```

}";

B.5 系统管理服务接口

B.5.1 查询应用状态的交互数据定义

B.5.1.1 应用向平台查询应用状态的交互数据定义参见表 B.16。

表 B.16 查询应用状态的交互数据定义

属性	含义	值	含义
dataclass	*数据种类	appstate	请求应用状态
datatype	*数据类型	curctxstate	查询自身的应用所处态
		curappstate	查询自身的应用状态
		appstate	查询指定某个节点某个应用的状态
optype	*操作类型	query	查询
appname	*应用名称	—	具体的应用名称(当datatype为appstate时需要填写此项，不填写时默认是查询自身应用)
nodename	*节点名称	—	具体的节点名称(当datatype为appstate时需要填写此项，不填写时默认是查询自身应用)
context	*应用态	—	int型值，具体的应用态(当datatype为appstate时需要填写此项，不填写时默认是自身应用的态)
data	返回查询的结果	参考具体数据类型的说明	curctxstate的请求返回值： 1.实时态 2.研究态 4.测试态 5.反演态 curappstate的请求返回值： 1. 值班 2. 热备 appstate的请求返回值： 1. 值班 2. 热备 3. 退出 4. 故障
result	返回信息功能码	功能码	信息功能码定义参见表B.12

B.5.1.2 应用向平台查询本身应用的所处态示例如下：

```
std::string sInitJSON = "{
                        "dataclass": "appstate",
                        "datatype": "curctxstate",
                        "optype": "query"
                        }";
```

B.5.1.3 平台向应用返回结果示例如下：

```
std::string sReturnJSON = "{
                          "dataclass": "appstate",
                          "datatype": "curctxstate",
                          "optype": "query",
                          "data": 1,
                          "result": 0
                          }";
```

B.5.1.4 应用向平台查询本身应用的当前状态示例如下：

```
std::string sInitJSON = "{
```

```
"dataclass": "appstate",
"datatype": "curappstate",
"optype": "query"
}";
```

B.5.1.5　平台向应用返回结果示例如下：

```
std::string sReturnJSON = "{
    "dataclass": "appstate",
    "datatype": "curappstate",
    "optype": "query",
    "data": 1,
    "result": 0
    }";
```

B.5.1.6　应用向平台查询指定某个节点某个应用的状态示例如下：

```
std::string sInitJSON = "{
"dataclass": "appstate",
"datatype": "appstate",
"optype": "query",
"appname": "scada",
"nodename": "scada1",
"context": 1
}";
```

B.5.1.7　平台向应用返回结果示例如下：

```
std::string sReturnJSON = "{
    "dataclass": "appstate",
    "datatype": "appstate",
    "optype": "query",        "appname": "scada",
    "nodename": "scada1",
    "context": 1,
    "data": 1,
    "result": 0
    }";
```

B.5.2　查询应用所属主机或备机的交互数据定义

B.5.2.1　应用向平台查询应用所属主机或备机的交互数据定义参见表 B.17。

表 B.17　查询应用的所属主机或备机节点名的交互数据定义

属性	含义	值	含义
dataclass	*数据种类	appnodestate	请求应用所属主机或备机节点
datatype	*数据类型	nodename	主机或备机节点名称
optype	*操作类型	query	查询
appname	*应用名称	—	具体的应用名称
context	*应用态	—	int型值，具体的应用态（可选项，不填写时默认是实时态，即为1）
appstatus	*应用状态	—	int 型值，具体的应用运行状态： 1. 主机 2. 备机
data	返回查询的结果	—	应用所属的主机或备机节点名称（多个节点名之间用逗号分隔）
result	返回信息功能码	功能码	信息功能码定义参见表B.12

B.5.2.2　应用向平台查询应用所属主机示例如下：

```
std::string sInitJSON = "{
                        "dataclass": "appnodestate",
                        "datatype": "nodename",
                        "optype": "query",
                        "appname": "scada",
                        "context": 1,
                        "appstatus": 1
                        }";
```

B.5.2.3　平台向应用返回结果示例如下：

```
std::string sReturnJSON = "{
                            "dataclass": "appnodestate",
                            "datatype": "nodename",
                            "optype": "query",
                            "appname": "scada",
                            "context": 1,
                            "appstatus": 1,
                            "data": "scada1",
                            "result": 0
                            }";
```

B.5.2.4　应用向平台查询应用的所属备机示例如下：

```
std::string sInitJSON = "{
                        "dataclass": "appnodestate",
                        "datatype": "nodename",
                        "optype": "query",
                        "appname": "scada",
                        "context": 1,
                        "appstatus": 2
                        }";
```

B.5.2.5　平台向应用返回结果示例如下：

```
std::string sReturnJSON = "{
                            "dataclass": "appnodestate",
                            "datatype": "nodename",
                            "optype": "query",
                            "appname": "scada",
                            "context": 1,
                            "appstatus": 2,
                            "data": "scada2,scada3,scada4",
                            "result": 0
                            }";
```

B.5.3　查询节点状态的交互数据定义

B.5.3.1　应用向平台查询节点状态的交互数据定义参见表 B.18。

表 B.18　查询节点状态的交互数据定义

属性	含义	值	含义
dataclass	*数据种类	nodestate	请求节点状态
datatype	*数据类型	nodestate	节点状态
optype	*操作类型	query	查询
nodename	*节点名称	—	具体的节点名称

data	返回查询的结果	—	int 型值，请求节点状态返回值： 1，运行 2，退出
result	返回信息功能码	功能码	信息功能码定义参见表B.12

B. 5. 3. 2　应用向平台查询指定节点的状态示例如下：

```
std::string sInitJSON = "{
                        "dataclass": "nodestate",
                        "datatype": "nodestate",
                        "optype": "query",
                        "nodename": "scada1"
                        }";
```

B. 5. 3. 3　平台向应用返回结果示例如下：

```
std::string sReturnJSON = "{
                          "dataclass": "nodestate",
                          "datatype": "nodestate",
                          "optype": "query",
                          "nodename": "scada1",
                          "data": 1,
                          "result": 0
                          }";
```

B. 5. 4　查询某一节点所有应用状态的交互数据定义

B. 5. 4. 1　应用向平台查询某一节点所有应用状态的交互数据定义参见表 B.19。

表 B. 19　查询应用的所属主机或备机节点名的交互数据定义

属性	含义	值	含义
dataclass	*数据种类	nodeappstate	请求某一节点所有应用状态
datatype	*数据种类	nodeappstate	某一节点所有应用状态
optype	*操作类型	query	查询
nodename	*节点名称	—	具体的节点名称（可选项，不填写时默认是自身节点）
context	*应用态	—	int型值，具体的应用态（可选项，不填写时默认是实时态，即为1）
data	返回查询的结果	数组类型	包含多个应用名和对应应用状态组合的数组： { "appname": "app1", "appstate": 1 }, { "appname": "app2", "appstate": 1 }, ...
result	返回信息功能码	功能码	信息功能码定义参见表B.12

B. 5. 4. 2　应用向平台查询应用的所属主机示例如下：

```
std::string sInitJSON = "{
                        "dataclass": "nodeappstate",
                        "datatype": "nodeappstate",
                        "optype": "query",
                        "nodename": "scada1",
                        "context": 1
```

```
}";
```

B.5.4.3　平台向应用返回结果示例如下：

```
std::string sReturnJSON = "{
                          "dataclass": "nodeappstate",
                          "datatype": "nodeappstate",
                          "optype": "query",
                          "nodename": "scada1",
                          "context": 1,
                          "data": [
                                  {
                                  "appname": "scada",
                                  "appstate": 1
                                  },
                                  {
                                  "appname": "fe",
                                  "appstate": 2
                                  }
                                  ],
                           "result": 0
                           }";
```

B.6　历史数据库服务接口

B.6.1　请求历史数据的交互数据定义

B.6.1.1　应用向平台请求历史数据的交互数据定义参见表 B.20。

表 B.20 请求历史数据的交互数据定义

属性	含义	值	含义
dataclass	*数据种类	hisdata	历史数据
datatype	*数据类型	measanalog	模拟量
optype	*操作类型	query	查询
field	*查询的数据结构	objid,objtype,value,datetime	特定选择字段，多个字段时用英文逗号分隔，历史采样信息结构定义参见表A.3
condition	*查询条件	objlist	对象索引号列表，多个对象索引号之间以英文逗号分隔
		starttime	历史查询的起始时间，时间数据使用yyyy-MM-dd HH:mm:ss.SSS时间格式
		endtime	历史查询的结束时间，时间数据使用yyyy-MM-dd HH:mm:ss.SSS时间格式
		interval	取值间隔，秒级整数。300代表5分钟取一个值。为0则表示5分钟。若取数间隔内存在多条采样记录，以第一条为准
		其它条件项	可根据具体场景约定扩展（可选）
result	返回信息功能码	功能码	信息功能码定义参见表B.12
data	返回的数据结果	JSON对象数组	返回的数据内容

B.6.1.2　应用向平台请求历史数据示例如下：

```
std::string sInputJSON = "{
```

```
"dataclass": "hisdata",
"datatype": "measanalog",
"optype": "query",
"field": "objid,objtype,value,datetime",
"condition": {
                "objlist": "4001",
                "starttime": "2020-06-01 01:00:00.000",
                "endtime": "2020-06-02 01:00:00.000",
                "interval": 300
                }
}";
```

B. 6. 1. 3　平台向应用返回历史数据结果示例如下：

```
std::string sOutputJSON = "{
                "dataclass": "hisdata",
                "datatype": "measanalog",
                "optype": "query",
                "field": "objid,objtype,value,datetime",
                "condition": {
                                "objlist": "4001",
                                "starttime": "2020-06-01 01:00:00.000",
                                "endtime": "2020-06-02 01:00:00.000",
                                "interval": 300
                                },
                "result": 0,
                "data": [
                        {
                        "objid": 4001,
                        "objtype": "measanalog",
                        "value": 220.26,
                        "datetime": "2020-06-01 01:00:00.000"
                        },
                        {
                        "objid": 4001,
                        "objtype": "measanalog",
                        "value": 219.18,
                        "datetime": "2020-06-01 03:00:00.000"
                        },
                        {
                        "objid": 4001,
                        "objtype": "measanalog",
                        "value": 221.63,
                        "datetime": "2020-06-01 08:00:00.000"
                        }
                        ]
                }";
```

B. 6. 2　请求历史统计数据的交互数据定义

B. 6. 2. 1　应用向平台请求历史统计数据的交互数据定义参见表B.21。

表 B. 21 请求历史统计数据的交互数据定义

属性	含义	值	含义
dataclass	*数据种类	hissta	历史统计数据
datatype	*数据类型	measanalog	模拟量
optype	*操作类型	query	查询

field	*查询的数据结构	objid,statimeunit,stadatetime, maxvalue,maxvaluetime,minvalue,minvaluetime,avgvalue	特定选择字段，多个字段时用英文逗号分隔，历史采样统计信息结构定义参见表B.5
condition	*查询条件	objlist	对象索引号列表，多个对象索引号之间以英文逗号分隔
		starttime	历史统计的起始时间，时间数据使用yyyy-MM-dd HH:mm:ss.SSS时间格式
		endtime	历史统计的结束时间，时间数据使用yyyy-MM-dd HH:mm:ss.SSS时间格式
		statimeunit	历史统计的时间单位，hour、day、month、year。hour将返回从starttime所在小时到endtime所在小时之间的每小时统计值。day将返回从starttime所在天到endtime所在天之间的每天统计值，month返回从starttime所在月到endtime所在月之间的每月的统计值，year同理
result	返回信息功能码	功能码	信息功能码定义参见表B.12
data	返回的数据结果	JSON对象数组	返回的数据内容

B. 6. 2. 2　应用向平台请求历史统计数据示例如下：

```
std::string sInputJSON = "{
                          "dataclass": "hissta",
                          "datatype": "measanalog",
                          "optype": "query",
                          "field":
                           "objid,statimeunit,stadatetime,maxvalue,maxvaluetime,minvalue,minvaluetime,av
                           gvalue",
                          "condition": {
                                       "objlist": "4001",
                                       "starttime": "2020-06-01 01:00:00.000",
                                       "endtime": "2020-06-02 23:59:00.000",
                                       "statimeunit": "day"
                                       }
                           }";
```

B. 6. 2. 3　平台向应用返回历史统计数据结果示例如下：

```
std::string sOutputJSON = "{
                           "dataclass": "hissta",
                           "datatype": "measanalog",
                           "optype": "query",
                           "field": "
                             objid,statimeunit,stadatetime,maxvalue,maxvaluetime,minvalue,minvaluetime,a
                             vgvalue",
                           "condition": {
                                        "objlist": "4001",
                                        "starttime": "2020-06-01 01:00:00.000",
                                        "endtime": "2020-06-02 23:59:00.000",
                                        "statimeunit": "day"
                                        },
                           "result": 0,
```

```
"data": [
        {
        "objid": 4001,
        "statimeunit":"day",
        "stadatetime": "2020-06-01 00:00:00.000",
        "maxvalue": 220.26,
        "maxvaluetime":"2020-06-01 03:54:43.000",
        "minvalue": 210.26,
        "minvaluetime":"2020-06-01 08:54:43.000",
        "avgvalue": 215.2
        },
        {
        "objid": 4001,
        "statimeunit":"day",
        "stadatetime": "2020-06-02 00:00:00.000",
        "maxvalue": 230.26,
        "maxvaluetime":"2020-06-02 17:33:50.000",
        "minvalue": 210.26,
        "minvaluetime":"2020-06-02 08:54:43.000",
        "avgvalue": 220.2
        }
        ]
}";
```


B.7 实时数据库服务接口

B.7.1 请求实时数据的交互数据定义

B.7.1.1 实时数据包括平台的实时数据和计量数据，应用向平台请求实时数据的交互数据定义参见表 B.22。

表 B.22 请求实时数据的交互数据定义

属性	含义	值	含义
dataclass	*数据种类	realdata	实时数据
datatype	*数据类型	subcontrolarea	区域
		substation	厂站
		basevoltage	电压类型
		bay	间隔
		breaker	断路器
		disconnector	刀闸
		grounddisconnector	接地刀闸
		compensator_p	并联补偿器
		compensator_s	串联补偿器
		powertransformer	变压器
		transformerwinding	变压器绕组
		energyconsumer	负荷
		acline	线路
		aclinesegment	交流线段
		aclineend	线段端点
		busbarsection	母线
		cntrl_ied	二次基本信息
		cntrl_ied_ldevice	二次定值区号
		measanalog	主设备遥测
		measpoint	主设备遥信
		auxiliary_device	辅助设备
		auxiliary_yx	辅助设备遥信

		auxiliary_yc	辅助设备遥测
		relaysig	保护信号
		mutualinductor	互感器
		arcsuppressioncoil	消弧线圈
		lightningarrester	避雷器
		electriccub	网门
		groundstake	接地桩线
		substation_area	变电站区域
		substation_cabinet	变电站屏柜
		sys_table_info	表信息
		sys_column_info	域信息
		sys_menu_info	菜单定义
		token_define	标志牌定义
		typedef	类型定义
optype	*操作类型	query	查询
		subscribe_cycle	周期订阅
		unsubscribe_cycle	取消周期订阅 1）应用取消订阅后将取消掉某一种数据类型（datatype）的所有已订阅请求 2）取消订阅时不需要发送field和condition信息，返回结果无data数据
		subscribe_emergency	触发订阅
		unsubscribe_emergency	取消触发订阅 1）应用取消订阅后将取消掉某一种数据类型（datatype）的所有已订阅请求 2）取消订阅时不需要发送field和condition信息，返回结果无data数据
field	*查询的数据结构	id,name,…	特定选择字段，多个字段时用英文逗号分隔
condition	*查询条件 （无查询条件时为非必填项）	SQL语句的where条件内容	SQL语句的where查询条件，应仅限定于当前数据表的单表直接查询条件，禁止条件中嵌套独立sql语句的间接查询
appname	查询的实时库所在的应用	scada	可选字段，无该字段则默认读取scada应用
context	查询的实时库所在应用的态	realtime	可选字段，无该字段则默认读取实时态
result	返回信息功能码	功能码	信息功能码定义参见表B.12
data	返回的数据结果	JSON对象数组	返回的请求字段的数据记录内容

B. 7. 1. 2　主动查询

B. 7. 1. 2. 1　应用主动向平台请求模拟量实时数据示例如下：

```
std::string sInputJSON = "{
                          "dataclass": "realdata",
                          "datatype": "measanalog",
                          "optype": "query",
                          "field": "id,name,st_id,value",
                          "condition": "st_id=612345678912343 and name like '%电压'"
                          }";
```

B. 7. 1. 2. 2　平台向应用返回模拟量实时数据结果示例如下：

```
std::string sOutputJSON = "{
                          "dataclass": "realdata",
                          "datatype": "measanalog",
                          "optype": "query",
                          "field": "id,name,st_id,value",
                          "condition": "st_id=612345678912343 and name like '%电压'",
                          "result": 0,
                          "data": [
                                  {
                                  "id": 112347912548871,
                                  "name": "A 相电压",
                                  "st_id": 612345678912343,
                                  "value": 220.156
                                  },
                                  {
                                  "id": 112347912548872,
                                  "name": "B 相电压",
                                  "st_id": 612345678912343,
                                  "value": 220.456                },
                                  {
                                  "id": 112347912548873,
                                  "name": "C 相电压",
                                  "st_id": 612345678912343,
                                  "value": 220.256
                                  }
                                  ]
                           }";
```

B.7.1.3　周期通知与触发通知

B.7.1.3.1　应用向平台周期订阅模拟量实时数据示例如下：

```
std::string sInputJSON = "{
                         "dataclass": "realdata",
                         "datatype": "measanalog",
                         "optype": "subscribe_cycle",
                         "field": "id,value,qual",
                         "condition": "st_id=612345678912343 and description like '%电压'"
                         }";
```

B.7.1.3.2　平台在订阅后直接向应用返回周期订阅模拟量实时数据的订阅结果示例如下：

```
std::string sOutputJSON = "{
                           "dataclass": "realdata",
                           "datatype": "measanalog",
                           "optype": "subscribe_cycle",
                           "field": "id,value,quality",
                           "condition": "st_id=612345678912343 and description like '%电压'",
                           "result": 0
                           }";
```

B.7.1.3.3　平台向应用周期返回全部请求的模拟量实时数据结果示例如下：

```
std::string sOutputJSON = "{
                          "dataclass": "realdata",
                          "datatype": "measanalog",
                          "optype": "subscribe_cycle",
                          "field": "id,value,quality",
                          "condition": "st_id=612345678912343 and description like '%电压'",
```

```
                "result": 1,
                "data": [
                        {
                        "id": 112347912548871,
                        "value": 220.156,
                        "quality": 0
                        },
                        {
                        "id": 112347912548872,
                        "value": 220.456,
                        "quality": 0
                        },
                        {
                        "id": 112347912548873,
                        "value": 220.256,
                        "quality": 0
                        }
                        ]
                }";
```

B.7.1.3.4 平台在订阅后直接向应用返回触发订阅模拟量实时数据的订阅结果示例如下：

```
std::string sInputJSON = "{
                "dataclass": "realdata",
                "datatype": "measanalog",
                "optype": "subscribe_emergency",
                "field": "id,value,quality",
                "condition": "st_id=612345678912343 and description like '%电压'"
                }";
```

B.7.1.3.5 平台向应用返回触发订阅模拟量实时数据的订阅结果示例如下：

```
std::string sOutputJSON = "{
                 "dataclass": "realdata",
                 "datatype": "measanalog",
                 "optype": "subscribe_emergency",
                 "field": "id,value,quality",
                 "condition": "st_id=612345678912343 and description like '%电压'",
                 "result": 0
                 }";
```

B.7.1.3.6 平台向应用触发通知满足条件的突变模拟量实时数据结果示例如下：

```
std::string sOutputJSON = "{
                "dataclass": "realdata",
                "datatype": "measanalog",
                "optype": "subscribe_emergency",
                "field": "id,value,quality",
                "condition": "st_id=612345678912343 and description like '%电压'",
                "result": 2,
                "data": [
                        {
                        "id": 112347912548872,
                        "value": 220.456,
                        "quality": 0
                        }
                        ]
```

```
                                }";
```

B.7.1.4 取消订阅

B.7.1.4.1 应用向平台取消已周期订阅的模拟量实时数据示例如下：

```
std::string sInputJSON = "{
                                "dataclass": "realdata",
                                "datatype": "measanalog",
                                "optype": "unsubscribe_cycle"
                                }";
```

B.7.1.4.2 平台向应用返回取消已周期订阅的模拟量实时数据结果示例如下：

```
std::string sOutputJSON = "{
                                 "dataclass": "realdata",
                                 "datatype": "measanalog",
                                 "optype": "unsubscribe_cycle",
                                 "result": 0
                                 }";
```

B.7.1.4.3 应用向平台取消已触发订阅的模拟量实时数据示例如下：

```
std::string sInputJSON = "{
                                "dataclass": "realdata",
                                "datatype": "measanalog",
                                "optype": "unsubscribe_emergency"
                                }";
```

B.7.1.4.4 平台向应用返回取消已触发订阅的模拟量实时数据结果示例如下：

```
std::string sOutputJSON = "{
                                 "dataclass": "realdata",
                                 "datatype": "measanalog",
                                 "optype": "unsubscribe_emergency",
                                 "result": 0
                                 }";
```

B.8 文件服务接口

B.8.1 请求文件数据的交互数据定义

B.8.1.1 应用向平台请求文件数据的交互数据定义参见表 B.23。

表 B.23 请求文件数据的交互数据定义

属性	含义	值	含义
dataclass	*数据种类	file	文件数据
datatype	*数据类型	file	文件信息
optype	*操作类型	query	请求文件
		queryfilelist	请求文件列表，返回的文件列表仅包含当前目录的所有文件和一级子目录信息，不进行文件传输
path	*目标存放路径	指定目录的路径	用于存放请求文件的绝对路径
condition	*请求条件（无查询条件时为非必填项）	querypath	请求文件路径（可为文件或目录相对路径，请求文件或文件列表时填写，可选，不填写则默认是文件服务器根目录）
result	返回信息功能码	功能码	信息功能码定义参见表B.12
data	返回的数据结果	文件列表	获得的文件名称（filename）JSON对象

			数组列表： 1） 请求文件时，若querypath为文件则返回的数据结果中filename直接填写文件名称，返回的文件存放在path字段指定的目录下； 2） 请求文件时，若querypath为目录，返回的数据结果仅为当前目录下的所有文件数据，filename直接填写文件名称，并将文件存放在path字段指定的目录下，且当前目录下的子目录及其子文件数据不作处理； 3） 请求文件列表queryfilelist时，返回的数据结果是包含路径信息的文件名称和当前一级的包含路径信息的子目录名称列表，若为子目录以正斜杠符号‘/’结尾。

B.8.1.2 应用向平台请求文件数据示例如下：

```
std::string sInputJSON = "{
                          "dataclass": "file",
                          "datatype": "file",
                          "optype": "query",
                          "path": "/users/ems/tmp",
                          "condition": {
                                        "querypath": "wavefiles"
                                        }
                          }";
```

B.8.1.3 平台向应用返回请求文件数据结果示例如下：

```
std::string sOutputJSON = "{
                           "dataclass": "file",
                           "datatype": "file",
                           "optype": "query",
                           "path":"/users/ems/tmp",
                           "condition": {
                                         "querypath": "wavefiles"
                                         },
                           "result": 0,
                           "data": [
                                    {
                                    "filename":"A_RCD_01_01_00279_00_00000001_000308012339917.cfg"
                                    },
                                    {
                                    "filename":"A_RCD_01_01_00279_00_00000001_000308012339917.dat"
                                    },
                                    {
                                    "filename":"A_RCD_01_01_00279_00_00000001_000308012339917.hdr"
                                    },
                                    {
                                    "filename":"A_RCD_01_01_00282_00_00000004_000308012732158.cfg"
                                    },
                                    {
                                    "filename":"A_RCD_01_01_00282_00_00000004_000308012732158.dat"
                                    },
                                    {
```

```
                    "filename":"A_RCD_01_01_00282_00_00000004_000308012732158.hdr"
                    }
                    ]
            }";
```

B.8.1.4　应用向平台请求文件列表数据示例如下：

```
std::string sInputJSON = "{
                    "dataclass": "file",
                    "datatype": "file",
                    "optype": "queryfilelist",
                    "condition": {
                                "querypath": "/"
                                }
                    }";
```

B.8.1.5　平台向应用返回请求文件列表数据结果（带自定义目录）示例如下：

```
std::string sOutputJSON = "{
                     "dataclass": "file",
                     "datatype": "file",
                     "optype": "queryfilelist",
                     "condition": {
                                 "querypath": "/"
                                 },
                     "result": 0,
                     "data": [
                              {
                              "filename":"substation_1.3_1.1_357237F5.scd.zip"
                              },
                              {
                              "filename":"substation_1.3_1.2_0988868A.scd.zip"
                              },
                              {
                              "filename":"substation_1.3_1.3_324B0A3A.scd.zip"
                              },
                              {
                              "filename":"wavefiles/"
                              }
                              ]
                     }";
```

B.8.2　存储文件数据的交互数据定义

B.8.2.1　应用向平台发送文件数据的交互数据定义参见表 B.24。

表 B.24　存储文件数据的交互数据定义

属性	含义	值	含义
dataclass	*数据种类	file	文件数据
datatype	*数据类型	file	文件信息
optype	*操作类型	save	保存文件
		delete	删除文件或文件夹
path	*源文件存放路径（optype是delete时为非必填项）	指定文件的路径	源文件当前存放的绝对路径
condition	*请求条件（无查询条件时为	dstpath	目标文件在文件服务器存放的相对路径（可为文件或目录路径。保存

	非必填项）		文件时，路径中的目录由平台负责创建，如果采用文件路径则文件名需与源文件名称相同。，可选，不填写则默认是文件服务器根目录）
result	返回信息功能码	功能码	信息功能码定义参见表B.12

B.8.2.2　应用通过平台向主站发送文件数据示例如下：

```
std::string sInputJSON = "{
                          "dataclass": "file",
                          "datatype": "file",
                          "optype": "save",
                          "path": "/users/ems/tmp/CF2201.cid",
                          "condition": {
                                        "dstpath": "GSY0002X"
                                        }
                          }";
```

B.8.2.3　平台向应用返回发送文件数据结果示例如下：

```
std::string sOutputJSON = "{
                           "dataclass": "file",
                           "datatype": "file",
                           "optype": "save",
                           "path": "/users/ems/tmp/CF2201.cid ",
                           "condition": {
                                         "dstpath": "GSY0002X"
                                         },
                           "result": 0
                           }";
```

B.9　告警服务接口

B.9.1　请求实时告警的交互数据定义

B.9.1.1　应用向平台请求实时告警数据的交互数据定义参见表 B.25。

表 B.25　请求实时告警的交互数据定义

属性	含义	值	含义
dataclass	*数据种类	realalarm	实时告警
datatype	*数据类型	alarm	告警信息
optype	*操作类型	subscribe_emergency	订阅
		unsubscribe_emergency	取消订阅 1）应用取消订阅后将取消掉某一种数据（dataclass）的所有已订阅请求 2）取消订阅时不需要发送field和condition信息，返回结果无data数据
field	*订阅的数据结构	alarmtime,content,alarmlevel,...	需要订阅的告警信息的字段，多个字段时用英文逗号分隔.具体告警信息结构参见附录B.2.2
condition	*请求条件（无查询条件时为非必填项）	subname	获取该厂站下的告警（可选）
		alarmlevel	获取该告警等级的告警（可选）
		bayname	获取该间隔下的告警（可选）
		equipmentname	获取该设备下的告警（可选）
		aornum	获取该责任区下的告警（可选）

result	返回信息功能码	功能码	信息功能码定义参见表B.12
data	返回的数据结果	JSON对象数组	返回的当前查询条件下的实时告警数据内容，具体告警信息结构参见附录B.2.2

B.9.1.2 应用向平台订阅实时告警示例如下：

```
std::string sInputJSON = "{
                          "dataclass": "realalarm",
                          "datatype":"alarm",
                          "optype": "subscribe_emergency",
                          "field": "objid,alarmlevel,content,alarmtime",
                          "condition": {
                                       "subname": "测试站",
                                       "alarmlevel": 4
                                       }
                          }";
```

B.9.1.3 平台在订阅后直接向应用返回触发订阅实时告警的订阅结果示例如下：

```
std::string sInputJSON = "{
                          "dataclass": "realalarm",
                          "datatype":"alarm",
                          "optype": "subscribe_emergency",
                          "field": "objid,alarmlevel,content,alarmtime",
                          "condition": {
                                       "subname": "测试站",
                                       "alarmlevel": 4
                                       },
                          "result": 0
                          }";
```

B.9.1.4 平台向应用触发返回实时告警结果示例如下：

```
std::string sOutputJSON = "{
                           "dataclass": "realalarm",
                           "datatype":"alarm",
                           "optype": "subscribe_emergency",
                           "field": "objid,alarmlevel,content,alarmtime",
                           "condition": {
                                        "subname": "测试站",
                                        "alarmlevel": 4
                                        },
                           "result": 2,
                           "data": [
                                   {
                                   "subname": "测试站",,
                                   "alarmlevel": 4,
                                   "content": "测试线路 1.WKH-892 馈线保护测控装置.检修压板开入  投入",
                                   "alarmtime":"2020-06-01 01:00:00.000"
                                   }
                                   ]
                           }";
```

B.9.1.5 应用向平台取消已订阅的实时告警示例如下：

```
std::string sInputJSON = "{
                          "dataclass": "realalarm",
```

```
"datatype":"alarm",
"optype": "unsubscribe_emergency"
}";
```

B.9.1.6　平台向应用返回取消已订阅的实时告警结果示例如下：

```
std::string sOutputJSON = "{
  "dataclass": "realalarm",
  "datatype":"alarm",
  "optype": "unsubscribe_emergency"
  "result": 0
  }";
```

B.9.2　应用发送告警的交互数据定义

B.9.2.1　应用向平台发送实时告警的交互数据定义参见表 B.26。

表 B.26　发送实时告警的交互数据定义

属性	含义	值	含义
dataclass	*数据种类	realalarm	实时告警
datatype	*数据类型	alarm	告警信息
optype	*操作类型	send	发送实时告警
data	*发送的实时告警数据	JSON对象数组	发送的实时告警数据内容，具体告警信息结构参见附录B.2.2
result	返回信息功能码	功能码	信息功能码定义参见表B.12

B.9.2.2　应用向平台发送实时告警示例如下：

```
std::string sInputJSON = "{
  "dataclass": "realalarm",
  "datatype": "alarm",
  "optype": "send",
  "data": [
          {
           "objid": 4001,
           "subname": "测试站",
           "content": "测试线路 1.WKH-892 馈线开关  合闸",
           "alarmgroup": "开关动作",
           "alarmitem": "合闸",
           "alarmtime":"2020-06-01 01:00:00.000"
           }
           ]
}";
```

B.9.2.3　平台向应用返回结果示例如下：

```
std::string sOutputJSON = "{
   "dataclass": "realalarm",
   "datatype": "alarm",
   "optype": "send",
   "data": [
           {
            "subname": "测试站",
            "content": "测试线路 1.WKH-892 馈线开关  合闸",
            "alarmgroup": "开关动作",
```

```
                "alarmitem": "合闸",
                "alarmtime":"2020-06-01 01:00:00.000"
                }
                ],
        "result": 0
        }";
```

B.9.3 请求历史告警的交互数据定义

B.9.3.1 应用向平台请求历史告警数据的交互数据定义参见表 B.27。

表 B.27 请求历史告警的交互数据定义

属性	含义	值	含义
dataclass	*数据种类	hisalarm	历史告警
datatype	*数据类型	alarm	告警信息
optype	*操作类型	query	查询
field	*查询的数据结构	alarmtime,content,alarmlevel,...	需要查询的告警信息的字段，多个字段时用英文逗号分隔.具体告警信息结构参见附录B.2.2
condition	*查询条件	starttime	历史查询的起始时间，时间数据使用yyyy-MM-dd HH:mm:ss.SSS时间格式
		endtime	历史查询的结束时间，时间数据使用yyyy-MM-dd HH:mm:ss.SSS时间格式
		alarmtype	告警类型，参照B4告警类型定义（可选）
		其它条件项	可根据具体场景约定扩展（可选）
result	返回信息功能码	功能码	信息功能码定义参见表B.12
data	返回的数据结果	JSON对象数组	返回满足查询条件定义的历史告警数据内容

B.9.3.2 应用向平台请求历史告警示例如下：

```
std::string sInputJSON = "{
                "dataclass": "hisalarm",
                "datatype":"alarm",
                "optype": "query",
                "field": "alarmtime,content,alarmlevel",
                "condition": {
                        "starttime": "2020-06-01 01:00:00.000",
                        "endtime": "2020-06-02 01:00:00.000",
                        "alarmtype": 5
                        }
        }";
```

B.9.3.3 平台向应用返回历史告警结果示例如下：

```
std::string sOutputJSON = "{
                "dataclass": "hisalarm",
                "datatype":"alarm",
                "optype": "query",
                "field": "alarmtime,content,alarmlevel",
                "condition": {
                        "starttime": "2020-06-01 01:00:00.000",
                        "endtime": "2020-06-02 01:00:00.000",
```

```
                    "alarmtype": 5
                    },
  "result": 0,
  "data": [
          {
          "alarmlevel": 4,
          "content": "测试线路 1.WKH-892 馈线保护测控装置.检修压板开入  投入",
          "alarmtime":"2020-06-01 01:00:00.000"
          },
          {
          "alarmlevel": 4,
          "content": "测试线路 1.WKH-893 馈线保护测控装置.检修压板开入  退出",
          "alarmtime":"2020-06-01 03:00:00.000"
          },
          {
          "alarmlevel": 4,
          "content": "测试线路 1.WKH-894 馈线保护测控装置.检修压板开入  投入",
          "alarmtime":"2020-06-01 08:00:00.000"
          }
          ]
}";
```

B.10　权限服务接口

B.10.1　请求用户登录的交互数据定义

B.10.1.1　应用向平台请求用户登录时平台接口或平台服务端应弹出统一的用户登录界面，界面包括用户名、密码、双因子的校验。应用向平台请求用户登录的交互数据定义参见表 B.28。

表 B.28　请求用户登录的交互数据定义

属性	含义	值	含义
dataclass	*数据种类	rightmanage	权限管理
datatype	*数据类型	userlogin	用户登录
optype	*操作类型	query	查询
username	返回的用户名	用户名	返回的权限校验通过的用户名称
userdesc	返回的用户描述	用户描述	返回的权限校验通过的用户描述
right	权限码列表	返回的已由应用管理分配给该应用的且该用户具备的权限码列表	返回的已由应用管理分配给该应用的且该用户具备的权限码列表，多个权限码使用逗号隔开，权限码定义参见表B.13
result	返回信息功能码	功能码	信息功能码定义参见表B.12

B.10.1.2　应用向平台请求校验的权限参数数据示例如下：

```
std::string sInitJSON = "{
                        "dataclass": "rightmanage",
                        "datatype": "userlogin",
                        "optype": "query"
                        }";
```

B.10.1.3　平台向应用返回结果示例如下：

```
std::string sReturnJSON = "{
                          "dataclass": "rightmanage",
                          "datatype": "userlogin",
                          "optype": "query",
                          "username": "zhangsan",
                          "userdesc": "张三",
                          "right": "3,4,5,6",
                          "result": 0
                          }";
```

B.10.2 请求权限校验的交互数据定义

B.10.2.1 应用向平台请求权限校验时平台接口或平台服务端应弹出统一的权限校验界面，界面包括操作人、密码、双因子的校验。应用向平台请求权限校验的交互数据定义参见表 B.29。

表 B.29 请求权限校验的交互数据定义

属性	含义	值	含义
dataclass	*数据种类	rightmanage	权限管理
datatype	*数据类型	rightcheck	权限校验
optype	*操作类型	query	查询
right	*请求校验的权限	权限码	权限码定义参见表B.13
username	返回的用户名	用户名	返回的权限校验通过的用户名称
userdesc	返回的用户描述	用户描述	返回的权限校验通过的用户描述
result	返回信息功能码	功能码	信息功能码定义参见表B.12

B.10.2.2 应用向平台请求校验的权限参数数据示例如下：

```
std::string sInitJSON = "{
                        "dataclass": "rightmanage",
                        "datatype": "rightcheck",
                        "optype": "query",
                        "right": 1
                        }";
```

B.10.2.3 平台向应用返回结果示例如下：

```
std::string sReturnJSON = "{
                          "dataclass": "rightmanage",
                          "datatype": "rightcheck",
                          "optype": "query",
                          "right": 1,
                          "username": "lisi",
                          "userdesc": "李四",
                          "result": 0
                          }";
```

B.10.3 请求权限描述的交互数据定义

B.10.3.1 应用向平台请求权限的描述信息的交互数据定义参见表 B.30。

表 B.30 请求权限校验的交互数据定义

属性	含义	值	含义
dataclass	*数据种类	rightmanage	权限管理
datatype	*数据类型	rightinfo	权限信息
optype	*操作类型	query	查询

right	*请求校验的权限	权限码	权限码定义参见表B.13
desc	返回的权限描述信息	权限描述信息	返回的权限描述信息
result	返回信息功能码	功能码	信息功能码定义参见表B.12

B.10.3.2 应用向平台请求权限描述信息的参数数据示例如下：

```
std::string sInitJSON = "{
                        "dataclass": "rightmanage",
                        "datatype": "rightinfo",
                        "optype": "query",
                        "right": 2
                        }";
```

B.10.3.3 平台向应用返回结果示例如下：

```
std::string sReturnJSON = "{
                          "dataclass": "rightmanage",
                          "datatype": "rightcheck",
                          "optype": "query",
                          "right": 2,
                          "desc": "模型维护权限",
                          "result": 0
                          }";
```

B.11 审计服务接口

B.11.1 存储审计日志的交互数据定义

B.11.1.1 应用向平台存储审计数据的交互数据定义参见表 B.31。

表 B.31 存储审计数据的交互数据定义

属性	含义	值	含义
dataclass	*数据种类	auditlog	审计日志
datatype	*数据类型	auditdata	审计数据
optype	*操作类型	save	保存
field	*发送的数据结构	username,datetime,auditsubject,auditobject,audittype,auditlevel,auditdesc	审计数据信息结构的所有字段，具体审计数据信息结构参见附录B.2.4
data	*发送的实时告警数据	JSON对象数组	存储的审计数据内容，具体审计数据信息结构参见附录B.2.4
result	返回信息功能码	功能码	信息功能码定义参见表B.12

B.11.1.2 应用向平台发送审计数据示例如下：

```
std::string sInputJSON = "{
                         "dataclass": "auditlog",
                         "datatype": "auditdata",
                         "optype": "save",
                         "field":                    "username,datetime,auditsubject,auditobject,
                    audittype,auditlevel,auditdesc",
                         "data": [
                                 {
                                 "username": "zhangsan",
                                 "datetime": "2020-06-01 01:00:00.000",
```

```
                "auditsubject": "模型维护工具",
                "auditobject": "遥信表记录",
                "audittype": 1,
                "auditlevel": 2,
                "auditdesc": "修改遥信表描述，将'重合闸出口软压板'改为'重合闸出口
                 软压板 1'"
               }
               ]
}";
```

B.11.1.2　平台向应用返回结果示例如下：

```
std::string sOutputJSON = "{
                "dataclass": "auditlog",
                "datatype": "auditdata",
                "optype": "save",
                "field": "username,datetime,auditsubject,auditobject,
                  audittype,auditlevel,auditdesc",
                "data": [
                        {
                        "username": "zhangsan",
                        "datetime": "2020-06-01 01:00:00.000",
                        "auditsubject": "模型维护工具",
                        "auditobject": "遥信表记录",
                        "audittype": 1,
                        "auditlevel": 2,
                        "auditdesc": "修改遥信表描述，将'重合闸出口软压板'改为'重合闸出
                          口软压板 1'"
                        }
                        ],
                "result": 0
                }";
```

B.12　操作控制服务接口

B.12.1　请求安全认证的交互数据定义

基础平台向第三方应用提供安全认证接口。第三方应用在请求操作控制服务前，应调用基础平台安全认证接口进行安全认证。安全认证校验通过后，基础平台接收到第三方应用的操作控制服务请求时，采用通信加密技术对交互数据进行加密传输。安全认证校验失败，基础平台应拒绝第三方应用的操作控制服务请求。

安全认证接口内部通过基础平台安全认证服务提供的统一接口进行安全认证校验（参见 4.10.3 章节安全认证服务功能及接口说明）。

第三方应用仅需在请求操作控制服务前调用基础平台安全认证接口进行安全认证，请求平台其他服务无需调用安全认证接口，即基础平台仅对操作控制服务的交互数据进行加密传输，其他服务交互数据无需加密控制。第三方应用调用安全认证接口认证通过后，在证书有效期内无需再重复调用认证接口。

B.12.1.1　应用向平台请求安全认证的交互数据定义参见表 B.32。

表 B.32　请求安全认证的交互数据定义

属性	含义	值	含义
dataclass	*数据种类	safetyauth	安全认证
modulename	*应用功能模块名	模块名	应用功能模块名提供给平台用以安全认证
firmname	*应用厂商名	厂商名	应用厂商名提供给平台用以安全认证

p12	*P12文件的保护口令	P12文件的保护口令	P12文件的保护口令提供给平台用以安全认证
result	返回信息功能码	功能码	信息功能码定义参见表B.12

B. 12. 1. 2　应用向平台请求安全认证示例：

```
std::string sInitJSON = "{
                        "dataclass": "safetyauth",
                        "modulename": "testapp",   "firmname": "nr",
                        "p12": "12345678"
                        }";
```

B. 12. 1. 3　平台向应用返回安全认证结果示例：

```
std::string sInitJSON = "{
                        "dataclass": "safetyauth",
                        "modulename": "testapp",
                        "firmname": "nr",
                        "p12": "12345678",
                        "result": 0
                        }";
```

B. 12. 2　控制请求的交互数据定义

B. 12. 2. 1　控制请求的交互数据定义见表 B.33。

表 B. 33　控制请求的交互数据定义

属性	含义	值	含义
dataclass	*数据种类	control	控制
datatype	*数据类型	output	常规遥控
		sync_output	同期遥控
		nov_output	无压遥控
		setpoint	遥调
		tap	挡位升降急停
optype	*操作类型	send	客户端发送遥控
ctrltime	*遥控时间	—	Unix时间，秒，uint32
ctrlmachine	*控制节点	—	控制节点
ctrluser	*控制人	—	控制人
guardermachine	*监护节点	—	监护节点
guarderuser	*监护人	—	监护人

data	请求控制的数据	JSON 对象
		
		{
		"id":4001,
		"value":1,
		"stage":"select/execute/cancel/direct",
		"retcode":"",
		"retmsg":""
		}
			请求控制对象为JSON键值对
			id:uint64
			value:float(遥控时取整:1：控合/升档；2:控分/降档；3：档位急停。
			遥调：取值)
			stage:string(select:选择
			execute:执行
			cancel:取消
			direct:直接控制
			select_back:选择返回
			execute_back:执行返回/直控返回
			cancel_back:取消返回)
			retcode:遥控请求返回码

			（详见附录A错误代码） retmsg:遥控请求返回信息
result	返回信息功能码	功能码	—

B. 12. 2. 2　应用主动向平台请求遥控选择示例如下：

```
std::string sInputJSON = "{
                          "dataclass": "control",
                          "datatype": "output",
                          "optype": "send",
                          "ctrltime": 1658841445,
                          "ctrlmachine": "",
                          "guardermachine": "",
                          "ctrluser": "",
                          "guarderuser": "",
                          "data": {
                                   "id": 4001,
                                   "value": 1,
                                   "stage": "select"
                                   }
                         }";
```

B. 12. 2. 3　平台向应用返回遥控结果示例如下：

```
std::string sOutputJSON = "{
                           "dataclass": "control",
                           "datatype": "output",
                           "optype": "send",
                           "ctrltime": 1658841445,
                           "ctrlmachine": "",
                           "guardermachine": "",
                           "ctrluser": "",
                           "guarderuser": "",
                           "data": {
                                    "id": 4001,
                                    "value": 1,
                                    "stage": "select_back",
                                    "retcode": "",
                                    "retmsg": ""
                                    },
                           "result": 0
                           }";
```

B. 12. 3 顺控操作调用的交互数据定义

B. 12. 3. 1　顺控操作调用的交互数据定义见表 B.34。

表 B. 34 顺控操作调用的交互数据定义

属性	含义	值	含义
dataclass	*数据种类	scontrol	顺控请求
datatype	*数据类型	ticket	顺控票
		ticketlist	顺控票列表
optype	*操作类型	get	操作票召唤
		preview	操作票预演
		previewcancel	预演取消
		execute	操作票开始执行
		executestep	操作票执行步骤

		pause	操作票暂停执行
		continue	操作票继续执行
		finish	操作票结束
		stop	操作票终止执行
		subscribe_emergency	顺控服务订阅
		unsubscribe_emergency	顺控服务取消订阅
condition	*条件	name	操作对象名。 票操作是票命:"厂站/电压等级.间隔_源态_目标态" 操作票列表操作时是间隔名
		backtype	返回类型，订阅可以按照backtype为单位订阅，取消以datatype为单位取消。 ticketcontent:票内容 previewresult:预演结果 executeresult:执行总结果 ticketlistcontent:操作票列表内容
		subid	厂站id
		bayid	对象id
		step	步骤，int
		ctrluser	操作人

data	返回的数据结果	JSON对象数组	根据condition和optype，返回格式不同，参见例子
			{
			“step”:1,
			“path”:”文件路径”,
			“retcode”:”子站返回码”,(必填)
			“retmsg”:””
			}

result	返回信息功能码	功能码	—

B. 12. 3. 2　应用订阅顺控票示例如下：

```
std::string sInputJSON = "{
                          "dataclass": "scontrol",
                          "datatype": "ticket",
                          "optype": "subscribe_emergency",
                          "condition": {
                                        "name": "东山变/110kV.103_运行_冷备用",
                                        "subid": "11111",
                                        "backtype": "ticketcontent"
                                        }
                          }";
```

B. 12. 3. 3　平台返回票订阅结果示例如下：

```
std::string sInputJSON = "{
                           "dataclass": "scontrol",
                           "datatype": "ticket",
                           "optype": "subscribe_emergency",
                           "condition": {
                                         "name": "东山变/110kV.103_运行_冷备用",
                                         "subid": "11111",
                                         "backtype": "ticketcontent"
```

```
                    },
"result": 0
}";
```

B.12.3.4 应用订阅顺控预演结果示例如下：

```
std::string sInputJSON = "{
"dataclass": "scontrol",
"datatype": "ticket",
"optype": "subscribe_emergency",
"condition": {
                    "name": "东山变/110kV.103_运行_冷备用",
                    "subid": "11111",
                    "backtype": "previewresult"
                    }
 }";
```

B.12.3.5 平台返回顺控预演结果订阅示例如下：

```
std::string sInputJSON = "{
"dataclass": "scontrol",
"datatype": "ticket",
"optype": "subscribe_emergency",
"condition": {
                    "name": "东山变/110kV.103_运行_冷备用",
                    "subid": "11111",
                    "backtype": "previewresult"
                    },
"result": 0
}";
```

B.12.3.6 应用订阅顺控执行结果示例如下：

```
std::string sInputJSON = "{
"dataclass": "scontrol",
"datatype": "ticket",
"optype": "subscribe_emergency",
"condition": {
                    "name": "东山变/110kV.103_运行_冷备用",
                    "subid": "11111",
                    "backtype": "executeresult"
                    }
}";
```

B.12.3.7 平台返回顺控执行结果订阅示例如下：

```
std::string sInputJSON = "{
"dataclass": "scontrol",
"datatype": "ticket",
"optype": "subscribe_emergency",
"condition": {
                    "name": "东山变/110kV.103_运行_冷备用",
                    "subid": "11111",
                    "backtype": "executeresult"
                    },
"result": 0
}";
```

B.12.3.8 应用取消订阅顺控示例（取消订阅所有）如下：

```
std::string sInputJSON = "{
                        "dataclass": "scontrol",
                        "datatype": "ticket",
                        "optype": "unsubscribe_emergency",
                        "condition": {
                                "name": "东山变/110kV.103_运行_冷备用"
                                }
                        }";
```

B.12.3.9 平台返回顺控取消订阅示例如下：

```
std::string sInputJSON = "{
                        "dataclass": "scontrol",
                        "datatype": "ticket",
                        "optype": "unsubscribe_emergency",
                        "condition": {
                                "name": "东山变/110kV.103_运行_冷备用"
                                },
                        "result": 0
                        }";
```

B.12.3.10 应用发送召票命令示例如下：

```
std::string sInputJSON = "{
                        "dataclass": "scontrol",
                        "datatype": "ticket",
                        "optype": "get",
                        "condition": {
                                "name": "东山变/110kV.103_运行_冷备用",
                                "subid": "111",
                                "bayid": "111",
                                "step": 0,
                                "ctrluser": "ctrluser"
                                }
                        }";
```

B.12.3.11 平台返回召票激活示例如下：

```
std::string sInputJSON = "{
                        "dataclass": "scontrol",
                        "datatype": "ticket",
                        "optype": "get",
                        "condition": {
                                "name": "东山变/110kV.103_运行_冷备用",
                                "subid": "111",
                                "bayid": "111",
                                "step": 0,
                                "ctrluser": "ctrluser"
                                },
                        "data": {
                                "retcode": "0",
                                "retmsg": "成功"

                                },
                        "result": 0
                        }";
```

B.12.3.12　平台返回票内容示例如下：

```
std::string sInputJSON = "{
                          "dataclass": "scontrol",
                          "datatype": "ticket",
                          "optype": "subscribe_emergency",
                          "condition": {
                                       "name": "东山变/110kV.103_运行_冷备用",
                                       "subid": "111",
                                       "backtype": "ticketcontent"
                                       },
                           "data": {
                                    "path": "票文件路径",
                                    "retcode": "0",
                                    "retmsg": "成功"
                                    },
                           "result": 0
                           }";
```

B.12.3.13　应用发送预演命令示例如下：

```
std::string sInputJSON = "{
                          "dataclass": "scontrol",
                          "datatype": "ticket",
                          "optype": "preview",
                          "condition": {
                                       "name": "东山变/110kV.103_运行_冷备用",
                                       "subid": "111",
                                       "bayid": "",
                                       "step": 0,
                                       "ctrluser": "ctrluser"
                                       }
                          }";
```

B.12.3.14　平台返回预演激活示例如下：

```
std::string sInputJSON = "{
                          "dataclass": "scontrol",
                          "datatype": "ticket",
                          "optype": "preview",
                          "condition": {
                                       "name": "东山变/110kV.103_运行_冷备用",
                                       "subid": "111",
                                       "bayid": "",
                                       "step": 0,
                                       "ctrluser": "ctrluser"
                                       },
                          "data": {
                                   "retcode": "0",
                                   "retmsg": "成功"

                                   },
                          "result":0
                          }";
```

B.12.3.15　平台返回预演结果示例如下：

```
std::string sInputJSON = "{
                          "dataclass": "scontrol",
```

```
"datatype": "ticket",
"optype": "subscribe_emergency",
"condition": {
                "name": "东山变/110kV.103_运行_冷备用",
                "subid": "111",
                "backtype": "previewresult"
                },
"data": {
          "step": 0,
          "retcode": "0",
       "retmsg": "成功"       },
"result": 0
}";
```

B. 12. 3. 16　应用发送预演取消命令示例如下：

```
std::string sInputJSON = "{
"dataclass": "scontrol",
"datatype": "ticket",
"optype": "previewcancel",
"condition": {
                "name": "东山变/110kV.103_运行_冷备用",
                "subid": "111",
                "bayid": "",
                "step": 0,
                "ctrluser": "ctrluser"
                }
 }";
```

B. 12. 3. 17　平台返回预演取消激活示例如下：

```
std::string sInputJSON = "{
"dataclass": "scontrol",
"datatype": "ticket",
"optype": "previewcancel",
"condition": {
                "name": "东山变/110kV.103_运行_冷备用",
                "subid": "111 变",
                "bayid": "",
                "step": 0,
                "ctrluser": "ctrluser"
                },
"data": {
          "retcode": "0",
          "retmsg": "成功"

          },
"result":0
}";
```

B. 12. 3. 18　应用发送执行命令示例如下：

```
std::string sInputJSON = "{
"dataclass": "scontrol",
"datatype": "ticket",
"optype": "execute",
"condition": {
```

```
                                "name": "东山变/110kV.103_运行_冷备用",
                                "subid": "111",
                                "bayid": "",
                                "step": 0,
                                "ctrluser": "ctrluser"
                                }
                        }";
```

B. 12. 3. 19　平台返回执行激活示例如下：

```
std::string sInputJSON = "{
                        "dataclass": "scontrol",
                        "datatype": "ticket",
                        "optype": "execute",
                        "condition": {
                                "name": "东山变/110kV.103_运行_冷备用",
                                "subid": "111",
                                "bayid": "",
                                "step": 0,
                                "ctrluser": "ctrluser"
                                },
                        "data": {
                                "retcode": "0",
                                "retmsg": "成功"

                                },
                        "result":0
                        }";
```

B. 12. 3. 20　应用发送执行某一步骤命令：

```
std::string sInputJSON = "{
                        "dataclass": "scontrol",
                        "datatype": "ticket",
                        "optype": "executestep",
                        "condition": {
                                "name": "东山变/110kV.103_运行_冷备用",
                                "subid": "111",
                                "bayid": "111",
                                "step": 1,
                                "ctrluser": "ctrluser"
                                }
                        }";
```

B. 12. 3. 21　平台返回某一步骤执行结果示例如下：

```
std::string sInputJSON = "{
                        "dataclass": "scontrol",
                        "datatype": "ticket",
                        "optype": "executestep",
                        "condition": {
                                "name": "东山变/110kV.103_运行_冷备用",
                                "subid": "111",
                                "bayid": "111",
                                "step": 1,
                                "ctrluser": "ctrluser"
                                },
                        "data": {
```

```
                "retcode": "0",
                "retmsg": "成功"
                },
        "result": 0
        }";
```

B. 12. 3. 22　平台返回总执行结果示例如下：

```
std::string sInputJSON = "{
        "dataclass": "scontrol",
        "datatype": "ticket",
        "optype": "subscribe_emergency",
        "condition": {
                "name": "东山变/110kV.103_运行_冷备用",
                "subid": "111",
                "backtype": "executeresult"
                },
        "data": {
                "retcode": "0",
                "retmsg": "成功"

                },
        "result":0
        }";
```

B. 12. 3. 23　应用发送执行暂停命令：

```
std::string sInputJSON = "{
        "dataclass": "scontrol",
        "datatype": "ticket",
        "optype": "pause",
        "condition": {
                "name": "东山变/110kV.103_运行_冷备用",
                "subid": "111",
                "bayid": "111",
                "step": 1,
                "ctrluser": "ctrluser"
                }
        }";
```

B. 12. 3. 24　平台返回执行暂停示例如下：

```
std::string sInputJSON = "{
        "dataclass": "scontrol",
        "datatype": "ticket",
        "optype": "pause",
        "condition": {
                "name": "东山变/110kV.103_运行_冷备用",
                "subid": "111",
                "bayid": "111",
                "step": 1,
                "ctrluser": "ctrluser"
                },
        "data": {
                "retcode": "0",
                "retmsg": "成功"

                },
```

```
                        "result":0
                        }";
```

B. 12. 3. 25　应用发送执行继续命令：

```
std::string sInputJSON = "{
                        "dataclass": "scontrol",
                        "datatype": "ticket",
                        "optype": "continue",
                        "condition": {
                                "name": "东山变/110kV.103_运行_冷备用",
                                "subid": "111",
                                "bayid": "111",
                                "step": 1,
                                "ctrluser": "ctrluser"
                                }
                        }";
```

B. 12. 3. 26　平台返回执行继续示例如下：

```
std::string sInputJSON = "{
                        "dataclass": "scontrol",
                        "datatype": "ticket",
                        "optype": "continue",
                        "condition": {
                                "name": "东山变/110kV.103_运行_冷备用",
                                "subid": "111",
                                "bayid": "111",
                                "step": 1,
                                "ctrluser": "ctrluser"
                                },
                        "data": {
                                "retcode": "0",
                                "retmsg": "成功"

                                },
                        "result": 0
}";
```

B. 12. 3. 27　应用发送终止命令：

```
std::string sInputJSON = "{
                        "dataclass": "scontrol",
                        "datatype": "ticket",
                        "optype": "stop",
                        "condition": {
                                "name": "东山变/110kV.103_运行_冷备用",
                                "subid": "111",
                                "bayid": "111",
                                "step": 1,
                                "ctrluser": "ctrluser"
                                }
                        }";
```

B. 12. 3. 28　平台返回终止示例如下：

```
std::string sInputJSON = "{
                        "dataclass": "scontrol",
                        "datatype": "ticket",
```

```
                        "optype": "stop",
                        "condition": {
                                "name": "东山变/110kV.103_运行_冷备用",
                                "subid": "111",
                                "bayid": "111",
                                "step": 1,
                                "ctrluser": "ctrluser"
                                },
                        "data": {
                                "retcode": "0",
                                "retmsg": "成功"

                                },
                        "result":0
                        }";
```

B. 12. 3. 29　应用订阅顺控操作票列表示例如下：

```
std::string sInputJSON = "{
                        "dataclass": "scontrol",
                        "datatype": "ticketlist",
                        "optype": "subscribe_emergency",
                        "condition": {
                                "name": "103",
                                "subid": "111",
                                "bayid": "111",
                                "backtype": "ticketlistcontent"
                                }
                        }";
```

B. 12. 3. 30　平台返回顺控操作票列表示例如下：

```
std::string sInputJSON = "{
                        "dataclass": "scontrol",
                        "datatype": "ticket",
                        "optype": "subscribe_emergency",
                        "condition": {
                                "name": "103",
                                "subid": "111",
                                "bayid": "111",
                                "backtype": "ticketlistcontent"
                                },
                        "result": 0
                        }";
```

B. 12. 3. 31　应用取消订阅顺控操作票列表示例如下：

```
std::string sInputJSON = "{
                        "dataclass": "scontrol",
                        "datatype": "ticketlist",
                        "optype": "unsubscribe_emergency",
                        "condition": {
                                "name": "103",
                                "subid": "111",
                                "bayid": "111",
                                "backtype": "ticketlistcontent"
                                }
                        }";
```

B. 12. 3. 32　平台返回取消顺控操作票列表示例如下：

```
std::string sInputJSON = "{
                          "dataclass": "scontrol",
                          "datatype": "ticket",
                          "optype": "unsubscribe_emergency",
                          "condition": {
                                        "name": "103",
                                        "subid": "111",
                                        "bayid": "111",
                                        "backtype": "ticketlistcontent"
                                        },
                          "result": 0
                          }";
```

B. 12. 3. 33　应用发送召唤操作票列表命令：

```
std::string sInputJSON = "{
                          "dataclass": "scontrol",
                          "datatype": "ticketlist",
                          "optype": "get",
                          "condition": {
                                        "name": "103",
                                        "subid": "111",
                                        "bayid": "111",
                                        "ctrluser": "ctrluser"
                                        }
                          }";
```

B. 12. 3. 34　平台返回操作票列表示例如下：

```
std::string sInputJSON = "{
                          "dataclass": "scontrol",
                          "datatype": "ticketlist",
                          "optype": "get",
                          "condition": {
                                        "name": "103",
                                        "subid": "111",
                                        "bayid": "111",
                                        "ctrluser": "ctrluser"
                                        },
                          "data": {
                                  "retcode": "0",
                                  "retmsg": "成功"
                                  },
                           "result": 0
                           }";
```

B. 12. 3. 35　平台返回操作票列表内容示例如下：

```
std::string sInputJSON = "{
                          "dataclass": "scontrol",
                          "datatype": "ticketlist",
                          "optype": "subscribe_emergency",
                          "condition": {
                                        "name": "103",
                                        "subid": "111",
                                        "bayid": "111",
```

```
                "ctrluser": "ctrluser"
                },
"data": {
        "path": "操作票列表文件路径",
        "retcode": "0",
        "retmsg": "成功"
        },
"result": 0
}";
```


B.13 自动对点服务接口

B.13.1 自动对点的交互数据定义

B.13.1.1 自动对点的交互数据定义见表 B.35。

表 B.35 自动对点的交互数据定义

属性	含义	值	含义
dataclass	*数据种类	pointscheck	信号自动对点
datatype	*数据类型	statusinput	遥信
		analoginput	遥测
		output	遥控（预置）
optype	*操作类型	start	开始对点，遥测遥信是开始模拟测试，模拟测试完成后返回；遥控是逐点下发预置（前一个预置结束后，再调用接口下发下一个）
		stop	终止对点
		query	查询点表信息
		getvalue	查询当前值
		subscribe_emergency	变化数据订阅,在对点开始前订阅，遥测遥信会返回对点过程中的变化数据，遥控会返回预置的报文结果数据，以data数组形式返回
		unsubscribe_emergency	取消变化数据订阅，将取消掉某一种数据类型（datatype）的所有已订阅请求，返回结果无data数据
condition	*请求条件	rtuname	采集站名称
		pointsnum	信号个数，当optype为query时，不填此项
		address	以逗号分隔多个地址，当optype为query时，不填此项，返回时此项值为空
		extend	ft_string类型，预留的扩展字段，应用可根据自身需求填写
result	返回信息功能码	功能码	信息功能码定义参见表B.12
data	订阅(optype为subscribe_emergency)后	type:类型包括analoginput, cos, soe	analoginput表示遥测. cos表示cos变位，

	返回的数据结构，为JSON对象数组		soe表示SOE变位，遥信对点时会返回cos和soe数据
		addr：点号	ft_int类型
		time:变化时间，当type为soe时，为SOE时间	时间数据使用yyyy-MM-dd HH:mm:ss.SSS时间格式
		value,当datatype为analoginput时为遥测数值	ft_double类型
		status, 当type为cos或soe时为信号状态	ft_int类型，0代表分位，1代表合位
		quality，质量位	ft_int类型，0代表正常，1代表被取代
	点表查询(optype为query)返回的数据结构，为JSON对象数组	addr:点号	ft_int类型
		name:中文描述	ft_string类型
	查询当前值(optype为getvalue)返回的数据结构，为JSON对象数组	addr:点号	ft_int类型
		value,当datatype为analoginput时为遥测数值	ft_double类型
		status, 当datatype为statusinput时为信号状态	ft_int类型，0代表分位，1代表合位

B.13.1.2　应用向平台下发开始对点示例如下:

```
std::string sInitJSON = "{
                        "dataclass": "pointscheck",
                        "datatype": "statusinput",
                        "optype": "start",
                        "condition": {
                                "rtuname": "110kV 测试站",
                                "pointsnum": 3,
                                "address": "3,4,5"
                                }
                        }";
```

B.13.1.3　平台向应用返回报文下发结果示例如下：

```
std::string sReturnJSON = "{
                        "dataclass": "pointscheck",
                        "datatype": "statusinput",
                        "optype": "start",
                        "condition": {
                                "rtuname": "110kV 测试站",
                                "pointsnum": 3,
                                "address": ""
                                },
                        "result": 0
                        }";
```

B.13.1.4　应用可通过实时数据访问的接口方式或订阅变化数据方式，获取变化数据。

B.13.1.4.1　应用向平台下发数据订阅示例如下:

```
std::string sInitJSON = "{
                        "dataclass": "pointscheck",
                        "datatype": "statusinput",
                        "optype": "subscribe_emergency",
```

```
                    "condition": {
                                "rtuname": "110kV 测试站",
                                "pointsnum": 3,
                                "address": "3,4,5"
                                }
                    }";
```

B. 13. 1. 4. 2　平台向应用发送数据订阅结果示例如下:

```
std::string sInitJSON = "{
                    "dataclass": "pointscheck",
                    "datatype": "statusinput",
                    "optype": "subscribe_emergency",
                    "condition": {
                                "rtuname": "110kV 测试站",
                                "pointsnum": 3,
                                "address": ""
                                },
                    "result": 0
                    }";
```

B. 13. 1. 4. 3　平台向应用返回变化数据结果示例如下:

```
 std::string sInitJSON = "{
                     "dataclass": "pointscheck",
                     "datatype": "statusinput",
                     "optype": "subscribe_emergency",
                     "condition": {
                                 "rtuname": "110kV 测试站",
                                 "pointsnum": 3,
                                 "address": ""
                                 },
                     "data": [
                            {
                            "type": "cos",
                            "addr": 3,
                            "time": "2022-08-01 08:11:12.223",
                            "status": 0,
                            "quality": 1
                            },
                            {
                            "type": "cos",
                            "addr": 4,
                            "time": "2022-08-01 08:11:12.323",
                            "status": 1,
                            "quality": 1
                            },
                            {
                            "type": "cos",
                            "addr": 5,
                            "time": "2022-08-01 08:11:12.423",
                            "status": 1,
                            "quality": 1
                            }
                            ],
                    "result": 2
```

```
}";
```

B. 13. 1. 4. 4　应用向平台下发取消数据订阅示例如下:

```
std::string sInitJSON = "{
                        "dataclass": "pointscheck",
                        "datatype": "statusinput",
                        "optype": "unsubscribe_emergency",
                        "condition": {
                                "rtuname": "110kV 测试站",
                                "pointsnum": 3,
                                "address": "3,4,5"
                                }
                        }";
```

B. 13. 1. 4. 5　平台向应用发送数据订阅结果示例如下:

```
std::string sInitJSON = "{
                        "dataclass": "pointscheck",
                        "datatype": "statusinput",
                        "optype": "unsubscribe_emergency",
                        "condition": {
                                "rtuname": "110kV 测试站",
                                "pointsnum": 3,
                                "address": ""
                                },
                        "result": 0
                        }";
```

B. 13. 1. 4. 6　应用向平台下发查询点表示例如下:

```
std::string sInitJSON = "{
                        "dataclass": "pointscheck",
                        "datatype": "statusinput",
                        "optype": "query",
                        "condition": {
                                "rtuname": "110kV 测试站"
                                }
                        }";
```

B. 13. 1. 4. 7　平台向应用返回查询点表示例如下:

```
std::string sInitJSON = "{
                        "dataclass": "pointscheck",
                        "datatype": "statusinput",
                        "optype": "query",
                        "condition": {
                                "rtuname":"110kV 测试站"
                                },
                        "data": [
                                {
                                "addr": 1,
                                "name": "220kV263 线路 1 测控_263 断路器"
                                },
                                {
                                "addr": 2,
                                "name": "220kV263 线路 1 测控_264 断路器"
                                }
                                ],
```

```
"result": 0
}";
```

B.13.1.4.8　应用向平台下发查询当前值示例如下:

```
std::string sInitJSON = "{
"dataclass": "pointscheck",
"datatype": "analoginput",
"optype": "getvalue",
"condition": {
"rtuname": "110kV 测试站",
"pointsnum": 2,
"address": "3,4"
}
}";
```

B.13.1.4.9　平台向应用返回当前值结果示例如下:

```
std::string sInitJSON = "{
"dataclass": "pointscheck",
"datatype": "analoginput",
"optype": "getvalue",
"condition": {
"rtuname": "110kV 测试站",
"pointsnum": 2,
"address": ""
},
"data": [
{
"addr": 3,
"value": 23.3
},
{
"addr": 4,
"value": 24.2
}
],
"result": 0
}";
```

B.14　数据调阅服务接口

B.14.1　文件调阅的交互数据定义

B.14.1.1　文件调阅的交互数据定义见表 B.36。

表 B.36 文件调阅的交互数据定义

属性	含义	值	含义
dataclass	*数据种类	dafile	站端文件数据
datatype	*数据类型	SCD	SCD文件
		RCD	网关机RCD文件
		CID	网关机降级CID模型文件
		COMTRADE	录波文件
		HISDATA	历史数据文件
		MEAS PROT	测控装置版本文件 保护装置版本文件

		FAULTRPT	故障分析报告
		SMARTALM	智能告警简报
		LOGRULES	防误规则文件
		customfile	APP自定义文件
optype	*操作类型	query	请求文件
		queryfilelist	请求文件列表，返回的文件列表仅包含当前目录的所有文件和一级子目录信息，不进行文件传输，非必填目标存放路径（path字段）
		send	请求下发文件，数据类型为RCD时起效，RCD文件的站端存放目录在网关机规范已定义
path	*文件存放路径	指定目录或文件的路径	1）请求文件时，path为存放请求文件的绝对目录路径 2）下发文件时，path为源文件当前存放的绝对文件路径
condition	*请求条件	rtuname	采集站名称
		starttime	起始时间（可选，仅用于请求COMTRADE或HISDATA文件，需要与endtime配合使用）
		endtime	结束时间（可选，仅用于请求COMTRADE或HISDATA文件，需要与starttime配合使用）
		iedname	站控层设备或二次设备装置名称（可选），iedname不填时，默认从综合应用主机返回
		querypath	请求文件路径（可为文件或目录相对路径，请求文件或文件列表时填写
result	返回信息功能码	功能码	信息功能码定义参见表B.12
data	返回的数据结果	文件列表	获得的文件名称（filename）JSON对象数组列表： 1）请求文件时，querypath为请求文件的相对文件路径，返回的数据结果中filename直接填写文件名称，返回的文件存放在path字段指定的目录下； 2）请求文件时，不支持将querypath设置为目录； 3）请求文件列表queryfilelist时，querypath为目录，返回的数据结果是包含路径信息的文件名称和当前一级的包含路径信息的子目录名称列表，若为子目录以正斜杠符号‘/’结尾。

B.14.1.2　应用向平台请求调阅文件数据示例如下：

```
std::string sInputJSON = "{
                          "dataclass": "dafile",
                          "datatype": "SCD",
                          "optype": "query",
                          "path": "/users/ems/tmp",
                          "condition": {
                                        "rtuname": "测试站"
                                        }
                         }";
```

B.14.1.3　平台向应用返回调阅请求文件数据结果示例如下：

```
std::string sOutputJSON = "{
                           "dataclass": "dafile",
                           "datatype": "SCD",
                           "optype": "query",
                           "path": "/users/ems/tmp",
                           "condition": {
                                         "rtuname": "测试站"
                                         },
```

```
"result": 0,
"data": [
        {
        "filename": "测试站_1.3_1.1_357237F5.scd.zip"
        }
        ]
}";
```

B.14.1.4 应用向平台请求下发文件数据示例如下：

```
std::string sInputJSON = "{
"dataclass": "dafile",
"datatype": "RCD",
"optype": "send",
"path": "/users/ems/tmp/测试站_1.3_1.1_357237F5.rcd.zip",
"condition": {
            "rtuname": "测试站"
            }
}";
```

B.14.1.5 平台向应用返回请求下发文件的数据结果示例如下：

```
std::string sOutputJSON = "{
 "dataclass": "dafile",
 "datatype": "RCD",
 "optype": "send",
 "path": "/users/ems/tmp/测试站_1.3_1.1_357237F5.rcd.zip",
 "condition": {
              "rtuname": "测试站"
              },
 "result": 0,
 }";
```

B.14.1.6 应用向平台请求调阅文件列表数据示例如下：

```
std::string sInputJSON = "{
"dataclass": "dafile",
"datatype": "SCD",
"optype": "queryfilelist"
}";
```

B.14.1.7 平台向应用返回请求调阅文件列表数据结果示例如下：

```
std::string sOutputJSON = "{
 "dataclass": "dafile",
 "datatype": "SCD",
 "optype": "queryfilelist",
 "result": 0,
 "data": [
        {
        "filename": "测试站_1.3_1.1_357237F5.scd.zip"
        },
        {
        "filename": "测试站_1.3_1.2_0988868A.scd.zip"
        },
        {
        "filename": "测试站_1.3_1.3_324B0A3A.scd.zip"
```

```
                    },
                    {
                    "filename": "测试站_1.3_1.4_324B0A3A.scd.zip"
                    }
                    ]
}";
```

B. 14. 2 动态数据集的交互数据定义

B. 14. 2. 1 设置动态数据集的交互数据定义见表 B.37。

表 B. 37 设置动态数据集的交互数据定义

属性	含义	值		含义
dataclass	*数据种类	datasetting		数据集
datatype	*数据类型	dynamic		动态数据集
optype	*操作类型	set		设置
		delete		释放
		get		读取
condition	*请求条件	rtuname		采集站名称
		iedname		装置名称
data	动态数据集合（JSON对象数组） 1、设置动态数据集时为创建的动态数据集名称和参引reference数组集合，参引为CMS格式，单独提供FC信息； 2、删除动态数据集时为删除动态数据集名称； 3、读取数据集数据时，如果 fcda 为空，表示读取该数据集全部数据；否则为读取特定数据点的数据。其中读取双位遥信时，使用 2Bit 定长位串定义，01 表示分状态，10 表示合状态	datasetReference		数据集参引
		fcda	reference	fcda参引
			fc	功能约束，状态值为ST，量测值为MX
			dataType	数据类型，数据属性DA的嵌套数据类型，包括数值、品质位和时标的类型组合
			value	数据值（字符串，读取时返回，设置时可空）
			q	数据品质（字符串，读取时返回，设置时可空）
			t	数据时标 (整型，精度到毫秒, 读取时返回，设置时可空)
result	返回信息功能码	功能码	—	信息功能码定义参见表B.12

B. 14. 2. 2 应用向平台请求设置动态数据集数据示例如下：

```
std::string sInputJSON = "{
                    "dataclass": "datasetting",
                    "datatype": "dynamic",
                    "optype": "set",
                    "condition": {
                                "rtuname": "测试站",
                                "iedname": "P_L2211B"
                                },
  "data": {
                                "datasetReference": "dsRelayDin1",
                                "fcda": [
```

```
                                        {
                                        "reference": "P_L2211BLD0/GGIO1.Ind1",
                                        "fc": "ST",
                                        "dataType": "{(stVal)Bstring2,(q)BVstring13,(t)Utctime}"
                                        },
                                        {
                                        "reference": "P_L2211BLD0/GGIO1.Ind2",
                                        "fc": "ST",
                                        "dataType": "{(stVal)Bstring2,(q)BVstring13,(t)Utctime}"
                                        },
                                        {
                                        "reference": "P_L2211BLD0/GGIO1.Ind2",
                                        "fc": "ST",
                                        "dataType": "{(stVal)Bstring2,(q)BVstring13,(t)Utctime}"
                                        }
                                        ]
                                }
                        }";
```

B. 14. 2. 3　平台向应用返回请求设置动态数据集数据示例如下：

```
std::string sInputJSON = "{
                        "dataclass": "datasetting",
                        "datatype": "dynamic",
                        "optype": "set",
                        "condition": {
                                "rtuname": "测试站",
                                "iedname": "P_L2211B"
                                },
  "data": {
                                "datasetReference": "dsRelayDin1",
                                "fcda": [
                                        {
                                        "reference": "P_L2211BLD0/GGIO1.ST.Ind1.stVal",
                                        "fc": "ST",
                                        "dataType": "{(stVal)Bstring2,(q)BVstring13,(t)Utctime}"
                                        },
                                        {
                                        "reference": "P_L2211BLD0/GGIO1.ST.Ind2.stVal",
                                        "fc": "ST",
                                        "dataType": "{(stVal)Bstring2,(q)BVstring13,(t)Utctime}"
                                        },
                                        {
                                        "reference": "P_L2211BLD0/GGIO1.ST.Alm1.stVal",
                                        "fc": "ST",
                                        "dataType": "{(stVal)Bstring2,(q)BVstring13,(t)Utctime}"
                                        }
                                        ]
                                },
                        "result": 0
                        }";
```

B. 14. 2. 4　应用向平台请求释放动态数据集数据示例如下：

```
std::string sInputJSON = "{
                        "dataclass": "datasetting",
                        "datatype": "dynamic",
```

```
"optype": "delete",
"condition": {
        "rtuname": "测试站",
        "iedname": "P_L2211B"
        },
"data": {
        "datasetReference": "dsRelayDin1"
        }
}";
```

B. 14. 2. 5　平台向应用返回释放设置动态数据集数据示例如下：

```
std::string sInputJSON = "{
"dataclass": "datasetting",
"datatype": "dynamic",
"optype": "delete",
"condition": {
        "rtuname": "测试站",
        "iedname": "P_L2211B"
        },
"data": {
        "datasetReference": "dsRelayDin1",
        },
"result": 0
}";
```

B. 14. 2. 6　应用向平台请求读动态数据集数据示例如下：

```
std::string sInputJSON = "{
 "dataclass": "datasetting",
 "datatype": "dynamic",
 "optype": "get",
 "condition": {
        "rtuname": "测试站",
        "iedname": "P_L2211B"
        },
 "data": {
        "datasetReference": "dsRelayDin1",
        "fcda": [
                {
                "reference": "P_L2211BLD0/GGIO1.Ind1",
                "fc": "ST",
                "dataType": "{(stVal)Bstring2,(q)BVstring13,(t)Utctime}"
                },
                {
                "reference": "P_L2211BLD0/GGIO1.Ind2",
                "fc": "ST",
                "dataType": "{(stVal)Bstring2,(q)BVstring13,(t)Utctime}"
                }
                ]
        }
}";
```

B. 14. 2. 7　平台向应用返回请求设置动态数据集数据示例如下：

```
std::string sInputJSON = "{
 "dataclass": "datasetting",
```

```
"datatype": "dynamic",
"optype": "get",
"condition": {
                "rtuname": "测试站",
                "iedname": "P_L2211B"
                },
"data": {
        "datasetReference": "dsRelayDin1",
        "fcda": [
                 {
                 "reference": "P_L2211BLD0/GGIO1.ST.Ind1.stVal",
                 "fc": "ST",
                 "dataType": "{(stVal)Bstring2,(q)BVstring13,(t)Utctime}",
                 "value": "01",
                 "q": "4096",
                 "t": 1676394061000
                 },
                 {
                 "reference": "P_L2211BLD0/GGIO1.ST.Ind2.stVal",
                 "fc": "ST",
                 "dataType": "{(stVal)Bstring2,(q)BVstring13,(t)Utctime}",
                 "value": "01",
                 "q": "4096",
                 "t": 1676394061000
                 }
                 ]
        },
"result": 0
}";
```


B.15 日志服务接口

B.15.1 存储应用日志的交互数据定义

B.15.1.1 应用向平台存储应用日志的交互数据定义参见表 B.38。

表 B.38 存储应用日志的交互数据定义

属性	含义	值	含义
dataclass	*数据种类	applog	应用日志
datatype	*数据类型	logdata	日志数据
optype	*操作类型	save	保存
data	*发送的日志数据	ctxname	应用场景
		appname	应用名称
		flag	对象标识： 0 - 表示记录对象为进程 1 - 表示记录对象为动态库
		objectname	对象名称：根据flag标识，为进程名或动态库库名
		logtime	日志生成的时间，格式参考
		loglevel	日志等级: LOG_DEBUG - 调试日志 LOG_INFO - 信息日志 LOG_NOTICE - 通知日志 LOG_WARNING - 告警日志

			LOG_ERR - 错误日志 LOG_ALERT - 报警日志 LOG_EMERG - 紧急日志
		logcontent	日志内容，不超过4096字符
result	返回信息功能码	功能码	信息功能码定义参见表B.12

B. 15. 1. 2　应用向平台发送审计数据示例如下：

```
std::string sInputJSON = "{
                          "dataclass": "applog",
                          "datatype": "logdata",
                          "optype": "save",
                          "data": [
                                  {
                                  "ctxname": "realtime",
                                  "appname": "scada",
                                  "flag": 0,
                                  "objectname": "scada-eps",
                                  "logtime": "2020-06-01 01:00:00.000",
                                  "loglevel": "LOG_ERR",
                                  "logcontent": "app test log"
                                  }
                                  ]
                           }";
```

B. 15. 1. 3　平台向应用返回结果示例如下：

```
std::string sOutputJSON = "{
                            "dataclass": "applog",
                            "datatype": "logdata",
                            "optype": "save",
                            "data": [
                                    {
                                    "ctxname": "realtime",
                                    "appname": "scada",
                                    "flag": 0,
                                    "objectname": "scada-eps",
                                    "logtime": "2020-06-01 01:00:00.000",
                                    "loglevel": "LOG_ERR",
                                    "logcontent": "app test log"
                                    }
                                    ],
                             "result": 0
                             }";
```

附 录 C
（资料性）
安全区Ⅳ服务化接口定义

安全区IV功能应用与基础平台之间的数据交互内容使用 JSON 格式字符串数据，交互数据分为请求接口输入参数数据和回调接口返回参数数据，交互时需要进行用户权限校验。

C.1 接口交互

C.1.1 总体要求

功能应用与基础平台之间使用统一的数据交互接口，用于读取基础平台的历史数据、实时数据、告警数据，及向基础平台存储审计日志数据，总体要求如下：

a) 基础平台应提供基于门户服务框架的统一的访问接口；
b) 交互接口参数采用 JSON 格式定义；
c) 数据交互接口应具备权限校验机制；
d) 数据交互应满足集控系统二次安全防护要求。

C.1.2 交互内容种类

功能应用与基础平台之间交互的数据内容种类如下：

a) 历史数据：包括历史采样、历史统计数据;
b) 实时数据：包括平台实时库数据;
c) 告警数据：包括历史告警数据;
d) 审计日志：包括应用运行所产生的日志信息。

C.1.3 交互操作类型

功能应用与基础平台之间的数据交互方式要求如下：

数据查询：功能应用通过主动召唤的方式请求查询需要的数据，采用 HTTPS 协议，使用 RESTful 风格进行开发，见图 C.1。

a) 查询的数据应支持查询条件过滤；
b) 查询的数据类型包括历史数据、实时数据、告警数据。

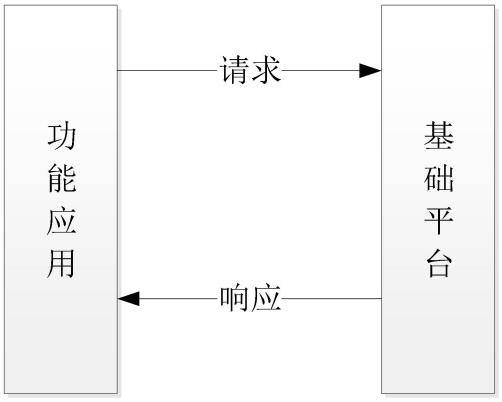


图 C.1 数据查询

C.1.4 交互格式定义

功能应用与基础平台之间的数据交互接口参数使用 JSON 格式字符串数据，交互数据分为请求接口输入参数数据和回调接口返回参数数据，具体参见接口定义。

为避免中文编码出现乱码问题，约定全局编码格式为 UTF-8。

C.2 接口定义

接口是由基础平台基于门户服务框架提供的一组服务，中间数据传输使用 JSON 格式字符串，用于通过数据信息结构读取和修改基础平台的数据，以及收发消息等。

接口风格参照 RESTful API 规范，以 Https URL 的形式体现资源和对应的接口方法。完整的接口方法由域名（IP 地址）、版本、路径（端点）、http 动词（GET 和 POST）、过滤信息（参数）、状态码、返回结果等组成。

1）URL

接口采用的 Https URL 形式：

接口 URL 中各组成部分的说明如下所列：

序号	URL 组成部分	说明
1	{host}:{port}	采用 http 协议访问接口所需的主机地址和端口
2	ccs	固定词，表示集控站
3	v{n}	表示版本号，如当前版本号为 v1
4	{endpoint}	表示路径（端点），用于标识具体的接口

采用本规范的每一个 URL 中，{endpoint}之前的部分（即 https://{host}:{port}/v{n}/ccs/）需根据集控站的情况进行组装。具体的接口内容在{endpoint}部分定义。

2）请求方法

支持采用 HTTP GET 或 HTTP POST 方法，具体参见接口定义。

3）消息头部

字段名	类型	值	是否必选	描述
Content-Type	String	权限认证交互：application/x-www-form-urlencoded 其它交互：application/json	是	发送的实体的 MIME 类型
Authorization	String	Bearer+空格+token 字符串	是	权限 Token 信息，内容为：token 类型 token

4）返回体为 JSON 格式对象，字段如下（C.3.1.2 Token 校验接口除外）。

字段名	类型	必要性	说明
code	String	必填	服务端提供业务返回码， 通过 5 位数字表示，00000 统一为业务请求成功，业务错误码则根据接口类型分别定义，前两位表示接口类型，后三位表示具体的错误，定义详见 C.3 接口说明
msg	String	必填	业务码描述信息
data	Object	选填	服务端的返回对象实体，默认值为 null

C.3 接口功能

C.3.1 权限认证管理

应通过 Token 对接口进行访问认证和安全管控，确保后端服务的安全性，保护客户信息资产安全。

C.3.1.1 Token 格式

Token 结构主要根据 RFC7519（https://tools.ietf.org/html/rfc7519）JSON WEB TOKEN 进行设计，JWT 一般被用来在身份提供者和服务提供者间传递被认证的用户身份信息，也可以增加一些额外的其它业务逻辑所必须的声明信息。其结构如下所示分为三部分，通过.进行连接：header.payload.signature。

header 为头部，通过 base64 进行编码，主要定义了 Token 的签名算法和类型，其结构如下：

typ	可选 （默认为 JWT）	Token 的类型
alg	可选（默认为 HS512）	签名算法类型

payload 为载荷，通过 base64 进行编码，内容定义如下：

iss	必填	Token 的签发者
sub	必填	Token 的所有者
exp	必填	Token 的过期时间（Long 类型时间戳），精确到秒
iat	可选	Token 的签发时间（Long 类型时间戳），精确到秒
nbf	可选	Token 的有效开始时间（Long 类型时间戳），精确到秒

原则上来说 JWT Token 在过期之前是无法失效的，但是可以通过存储在数据中间件中以进行过期校验和会话时长校验。

C.3.1.2 Token 校验

1) 接口定义

请求方法	POST
请求地址	auth-service/oauth/check_token
协议应用	https

2) 请求参数

请求参数类型为 application/x-www-form-urlencoded，内容如下：

属性	名称	类型	必要性	说明
token	令牌	String	必填	待校验的令牌

3) 返回数据

校验成功，响应体定义如下：

属性	名称	类型	必要性	说明
exp	access_token 有效期	Integer	必填	有效期时间戳，精度到秒
sub	被授权方	String	必填	一般就是当前用户 ID
iss	授权方	String	必填	ccs@nrec
user_name	当前用户名	String	必填	当前用户名
code	业务返回码	String	必填	00000
msg	业务码描述信息	String	必填	业务请求成功

返回成功示例如下：


```
{
"sub": "40a21660947c44a6a40030109214f313",
"user_name": "web",
"iss": "nrec_auth",
"exp": 1554864193,
"code": "00000",
"msg": "token 校验成功"
}
```


校验失败，响应体定义如下：

属性	名称	类型	说明
code	业务返回码	String	定义如下： 01001，校验 token 失败 01002，token 已过有效期 01003，缺少 token

msg	业务码描述信息	String	详见 code 说明

返回失败示例如下：


```
{
"code": "01001",
"msg": "token 校验失败"
}
```


C.3.1.3 基于 Token 获取用户信息

1) 接口定义

请求方法	GET
请求地址	per-service/platform/user-info
协议应用	https

2) 请求参数

消息头部包括类型、Token，应遵循 C.2 中消息头部定义。

3) 返回数据

返回数据遵循 C.2 中关于返回体的定义，业务返回码信息如下：

属性	名称	类型	说明
code	业务返回码	String	定义如下： 00000，业务请求成功 02001，用户不存在
msg	业务码描述信息	String	详见 code 说明

查询成功，data 内容定义如下：

属性	名称	类型	必要性	说明
userId	用户 ID	String	必填	
name	用户名称	String	必填	
nameCN	用户中文名	String	必填	

查询失败，data 值可不填，默认为 null。

返回成功示例如下：


```
{
"code": "00000",
"msg": "业务请求成功",
"data":
        {
        "userId": "39c97e381b5948c9b44a9b322ab954be",
        "name": "zhangsan",
        "nameCN": "张三"
        }
}
```


返回失败示例如下：


```
{
  "code": "02001",
  "msg": "用户不存在",
  "data": null
}
```

C.3.1.4 基于 Token 获取用户菜单权限信息

1）接口定义

请求方法	GET
请求地址	per-service/platform/menu
协议应用	https

2）请求参数

消息头部包括类型、Token，应遵循 C.2 中消息头部定义。

3)返回数据

返回数据遵循 C.2 中关于返回体的定义，业务返回码信息如下：

属性	名称	类型	说明
code	业务返回码	String	定义如下： 00000，业务请求成功 03001，用户不存在
msg	业务码描述信息	String	详见 code 说明

查询成功，data 内容定义为 List<Menu>，Menu 对象定义如下：

属性	名称	类型	必要性	说明
id	菜单 id	String	必填	
fid	菜单所属父菜单 id	String	必填	
ordernum	菜单排序	Integer	必填	
name	菜单名称	String	必填	
route	菜单路由	String	必填	

查询失败，data 值可不填，默认为 null。

返回成功示例如下：


```
{
"code": "00000",
"msg": "业务请求成功",
"data": [
        {
        "id": "0fac14ca29f3407a90eac34054f8f6b1",
        "fid": "19698adff79e45769b31c7305c332c84",
        "ordernum": 1,
        "name": "用户管理",
        "route": "/console/user"
        },
        {
        "id": "22af2941804bd0c877ae239b3cf3d0fa",
        "fid": "19698adff79e45769b31c7305c332c84",
        "ordernum": 2,
        "name": "资源管理",
        "route": "/console/resource"
        }
        ]
}
```


返回失败示例如下：

{
"code": "03001",
"msg": "用户不存在",
"data": null
}

C.3.1.5 基于 Token 获取用户功能权限信息

1) 接口定义

请求方法	GET
请求地址	per-service/platform/function
协议应用	https

2) 请求参数

消息头部包括类型、Token，应遵循 C.2 中消息头部定义。

3) 返回数据

返回数据遵循 C.2 中关于返回体的定义，业务返回码信息如下：

属性	名称	类型	说明
code	业务返回码	String	定义如下： 00000，业务请求成功 04001，用户不存在
msg	业务码描述信息	String	详见 code 说明

查询成功时，data 内容定义为 List<String>。

查询失败，data 值可不填，默认为 null。

返回成功示例如下：

{
"code": "00000",
"msg": "业务请求成功",
"data": ["function1", "function2", "function3"]
}

返回失败示例如下：

{
"code": "04001",
"msg": "用户不存在",
"data": null
}

C.3.2 历史数据服务接口

C.3.2.1 查询历史数据

1）接口定义

请求方法	POST
请求地址	hisdata-service/data/history
协议应用	https

2）请求参数

消息头部包括类型、Token，应遵循 C.2 中消息头部定义。

请求体（Request body）为 JSON 格式对象，其数据的定义如下：

属性	名称	类型	必要性	说明
keys	信号索引键	List<Integer>	必填	需查询的信号列表

startTime	开始时间	String	必填	开始时间（含），格式样例 "yyyy-MM-dd HH:mm:ss.SSS"
endTime	结束时间	String	必填	结束时间（含），格式样例 "yyyy-MM-dd HH:mm:ss.SSS"
interval	查询步长	Integer	必填	取值间隔，秒数，300 代表 5 分钟取一个值。为 0 则表示 5 分钟。若取数间隔内存在多条采样记录，以第一条为准

请求参数 JSON 示例:


```
{
"keys": [4001,4002],
"startTime": "2023-01-01 00:00:00.000",
"endTime": "2023-01-02 00:00:00.000",
"interval": 300
}
```


3）返回数据

返回数据遵循 C.2 中关于返回体的定义，业务返回码信息如下：

属性	名称	类型	说明
code	业务返回码	String	定义如下： 00000，业务请求成功 05001，查询信号数过多（大于 10000） 05002，查询时间区间过大（开始时间与结束时间相距大于 1 年）。如确有查大跨度时间的需求，可分多次小时间跨度查询 05003，预期结果条数过多（大于 100 万条。预期结果条数为推算而出，即信号点数×（结束时间-开始时间）/查询步长）。如确有查大批量数据的需求，可多次查询小时间跨度或者多次查询信号点子集 05004，信号索引键解析错误（未在模型中找到对应信号点） 05005，参数校验错误（开始时间大于结束时间、查询步长为负数等） 05006，内部执行错误
msg	业务码描述信息	String	详见 code 说明

查询成功时，其 data 内容定义为 ListHisKeyData>，HisKeyData 对象定义如下：

属性	名称	类型	必要性	说明
key	信号索引键	Integer	必填	信号的索引键
values	时刻与值的序列	List<HisData>	必填	采样值序列

HisData 对象定义如下：

属性	名称	类型	必要性	说明
value	值	Double	必填	采样值
time	时刻	String	必填	采样时刻，格式样例"yyyy-MM-dd HH:mm:ss.SSS"

查询失败，data 值可不填，默认为 null 。

返回数据成功示例:


```
{
"code": "00000",
"msg": "业务请求成功",
"data": [
```

```
        {
        "key": 4001,
        "values": [
                {
                "value": 223.1,
                "time": "2023-01-01 00:00:00.000"
                },
                {
                "value": 223.1,
                "time": "2023-01-01 00:05:00.000"
                }
                ]
        },
        {
        "key": 4002,
        "values": [
                {
                "value": 223.1,
                "time": "2023-01-01 00:00:00.000"
                },
                {
                "value": 223.1,
                "time": "2023-01-01 00:05:00.000"
                }
                ]
        }
        ]
}
```


返回数据失败示例：


```
{
"code": "05004",
"msg": "信号索引键解析错误",
"data": null
}
```


C.3.2.2 查询历史统计

1）接口定义

请求方法	POST
请求地址	hisdata-service/data/hisstatistics
协议应用	https

2）请求参数

消息头部包括类型、Token，应遵循 C.2 中消息头部定义。

请求体（Request body）为 JSON 格式对象，其数据的定义如下：

属性	名称	类型	必要性	说明
keys	信号索引键	List<Integer>	必填	需查询的信号列表
startTime	开始时间	String	必填	开始时间（含），格式样例 "yyyy-MM-dd HH:mm:ss.SSS"
endTime	结束时间	String	必填	结束时间（含），格式样例 "yyyy-MM-dd HH:mm:ss.SSS"
statimeunit	统计时间单位	String	必填	历史统计的时间单位，hour、day、month、year。hour 将返回从 starttime 所在小时到

				endtime 所在小时之间的每小时统计值。day 将返回从 starttime 所在天到 endtime 所在天之间的每天统计值，month 返回从 starttime 所在月到 endtime 所在月之间的每月的统计值，year 同理

请求参数 JSON 示例：

```
{
"keys": [4001,4002],
"startTime": "2023-01-01 00:00:00.000",
"endTime": "2023-01-02 23:59:59.000",
"statimeunit": "day"
}
```

3）返回数据

返回数据遵循 C.2 中关于返回体的定义，业务返回码信息如下：

属性	名称	类型	说明
code	业务返回码	String	定义如下： 00000，业务请求成功 06001，查询信号数过多（大于 10000） 06002，预期结果条数过多（大于 100 万条。预期结果条数为推算而出，即信号点数×（结束时间-开始时间）/统计时间单位）。如确有查大批量统计数据的需求，可多次查询小时间跨度或者多次查询信号点子集 06003，信号索引键解析错误（未在模型中找到对应信号点） 06004，参数校验错误（开始时间大于结束时间、统计时间单位不是 hour、day、month、year 之一等） 06005，内部执行错误
msg	业务码描述信息	String	详见 code 说明

查询成功时，其 data 内容定义为 List<HisKeySta>，HisKeySta 对象定义如下：

属性	名称	类型	必要性	说明
key	信号索引键	Integer	必填	信号的索引键
statimeunit	统计时间单位	String	必填	统计时间单位，同请求中的 statimeunit
values	统计值序列	List<HisSta>	必填	统计值序列

HisSta 对象定义如下：

属性	名称	类型	必要性	说明
stadatetime	统计周期起始时间	String	必填	统计周期起始时间，格式样例"yyyy-MM-dd HH:mm:ss.SSS"
maxvalue	最大值	Double	必填	统计周期内最大值
maxvaluetime	最大值出现时间	String	必填	统计周期内最大值出现的时刻，格式样例"yyyy-MM-dd HH:mm:ss.SSS"
minvalue	最小值	Double	必填	统计周期内最小值
minvaluetime	最小值出现时间	String	必填	统计周期内最小值出现的时刻，格式样例"yyyy-MM-dd HH:mm:ss.SSS"
avgvalue	平均值	Double	必填	统计周期内的平均值

查询失败，data 值可不填，默认为 null 。

返回数据成功示例:


```
{
"code": "00000",
"msg": "业务请求成功",
"data": [
        {
        "key": 4001,
        "statimeunit": "day",
        "values": [
                {
                "stadatetime": "2023-01-01 00:00:00.000",
                "maxvalue": 224,
                "maxvaluetime": "2023-01-01 13:21:06.000",
                "minvalue": 210,
                "minvaluetime": "2023-01-01 18:20:06.000",
                "avgvalue": 217
                },
                {
                "stadatetime": "2023-01-02 00:00:00.000",
                "maxvalue": 224,
                "maxvaluetime": "2023-01-02 13:21:06.000",
                "minvalue": 210,
                "minvaluetime": "2023-01-02 18:20:06.000",
                "avgvalue": 217
                }
                ]
          },
          {
          "key": 4002,
          "statimeunit": "day",
          "values": [
                  {
                  "stadatetime": "2023-01-01 00:00:00.000",
                  "maxvalue": 224,
                  "maxvaluetime": "2023-01-01 13:21:06.000",
                  "minvalue": 210,
                  "minvaluetime": "2023-01-01 18:20:06.000",
                  "avgvalue": 217
                  },
                  {
                  "stadatetime": "2023-01-02 00:00:00.000",
                  "maxvalue": 224,
                  "maxvaluetime": "2023-01-02 13:21:06.000",
                  "minvalue": 210,
                  "minvaluetime": "2023-01-02 18:20:06.000",
                  "avgvalue": 217
                  }
                  ]
        }
        ]
}
```


返回数据失败示例：


```
{
```

```
    "code": "06003",
    "msg": "信号索引键解析错误",
    "data": null
}
```

C.3.3 实时数据库服务接口

C.3.3.1 主动查询

1）接口定义

请求方法	POST
请求地址	realdata-service/data/realtime
协议应用	https

2）请求参数

消息头部包括类型、Token，应遵循 C.2 中消息头部定义。

请求体（Request body）为 JSON 格式对象，其数据的定义如下：

属性	名称	类型	必要性	说明
datatype	数据类型	String	必填	需查询的数据库表名
field	查询的数据结构	String	必填	特定选择字段，多个字段时用英文逗号分隔,id,name...
condition	查询条件	String	必填	SQL 语句的 where 查询条件，应仅限定于当前数据的单表直接查询条件，禁止条件中嵌套独立 sql 语句的间接查询，""表示没有查询条件
appname	查询的实时库所在的应用	String	可选	无该字段则默认读取 scada 应用
context	查询的实时库所在应用的态	String	可选	无该字段则默认读取实时态

请求参数 JSON 示例：

```
{
"datatype": "measanalog",
"field": "id,name,value",
"condition": "st_id=612345678912343 and name like '%电压'"
}
```

3）返回数据

返回数据遵循 C.2 中关于返回体的定义，业务返回码信息如下：

属性	名称	类型	说明
code	业务返回码	String	定义如下： 00000，业务请求成功 07001，field 字段错误 07002，查询超时
msg	业务码描述信息	String	详见 code 说明

查询成功时，其 data 内容定义如下：

属性	名称	类型	必要性	说明
fieldtype	字段类型说明	RetType	必填	
fielddata	查询数据	List<RetData>	必填	

RetType 对象针对查询字段返回相应类型说明（包括"String"、"Double"、"Integer"、"Boolean"、"DateTime"），属性数目与查询字段的数目保持一致，属性标识与查询字段标识保持一致，属性类型均为 String，值表示相应字段的数值类型，以 id,name,value 查询字段示例如下：

属性	类型	必要性	说明
id	String	必填	值表示查询数据 id 字段的类型，"Integer"
name	String	必填	值表示查询数据 name 字段的类型，"String"
value	String	必填	值表示查询数据 value 字段的类型，"Double"

RetData 对象是查询的数据，属性数目与查询字段的数目保持一致，属性标识与查询字段标识保持一致，属性类型与 RetType 中相应的属性值说明保持一致，以 id,name,value 查询字段示例如下：

属性	类型	必要性	说明
id	Integer	必填	
name	String	必填	
value	Double	必填	

查询失败，data 值可不填，默认为 null。

返回数据成功示例:


```
{
"code": "00000",
"msg": "业务请求成功",
"data": {
        "fieldtype": {
                      "id": "Integer",
                      "name": "String",
                      "value":"Double"
                      },
        "fielddata": [
                      {
                      "id": 112347912548871,
                      "name": "A 相电压",
                      "value": 220.156
                      },
                      {
                      "id": 112347912548872,
                      "name": "B 相电压",
                      "value": 220.456
                      }
                      ]
        }
}
```


返回数据失败示例：


```
{
    "code": "07001",
    "msg": "field 字段错误",
    "data": null
}
```


C.3.4 告警服务接口

C.3.4.1 查询历史告警

1）接口定义

请求方法	POST
请求地址	hisalarm-service/data/alarm
协议应用	https

2）请求参数

消息头部包括类型、Token，应遵循 C.2 中消息头部定义。

请求体（Request body）为 JSON 格式对象，其数据的定义如下：

属性	名称	类型	必要性	作用
field	查询的数据结构	String	必填	特定选择字段，多个字段时用英文逗号分隔
starttime	起始时间	String	必填	历史查询的起始时间，格式样例"yyyy-MM-dd HH:mm:ss.SSS"
endtime	结束时间	String	必填	历史查询的结束时间，格式样例"yyyy-MM-dd HH:mm:ss.SSS"
alarmtype	告警类型	Integer	可选	告警类型

请求参数 JSON 示例：

```
{
"field": "alarmtime,content,alarmlevel",
"starttime": "2020-06-01 01:00:00.000",
"endtime": "2020-06-02 01:00:00.000",
"alarmtype": 4
}
```

3）返回数据

返回数据遵循 C.2 中关于返回体的定义，业务返回码信息如下：

属性	名称	类型	说明
code	业务返回码	String	定义如下： 00000，业务请求成功 08001，field 字段错误 08002，查询超时
msg	业务码描述信息	String	详见 code 说明

查询成功时，其 data 内容定义如下：

属性	名称	类型	必要性	说明
fieldtype	字段类型说明	RetType	必填	
fielddata	查询数据	List<RetData>	必填	

RetType 对象针对查询字段返回相应类型说明（包括"String"、"Double"、"Integer"、"Boolean"、"DateTime"），属性数目与查询字段的数目保持一致，属性标识与查询字段标识保持一致，属性类型均为 String，值表示相应字段的数值类型，以 alarmtime,content,alarmlevel 查询字段示例如下：

属性	类型	必要性	说明
alarmtime	String	必填	值表示查询数据 id 字段的类型，"String"
content	String	必填	值表示查询数据 name 字段的类型，"String"
alarmlevel	String	必填	值表示查询数据 value 字段的类型，"Integer"

RetData 对象是查询的数据，属性数目与查询字段的数目保持一致，属性标识与查询字段标识保持一致，属性类型与 RetType 中相应的属性值说明保持一致，以 alarmtime,content,alarmlevel 查询字段示例如下：

属性	类型	必要性	说明
alarmtime	String	必填	
content	String	必填	
alarmlevel	Integer	必填	

查询失败，data 值可不填，默认为 null。

返回数据成功示例:


```
{
"code": "00000",
"msg": "业务请求成功",
"data": {
        "fieldtype": {
                        "alarmtime": "String",
                        "content": "String",
                        "alarmlevel": "Integer"
                        },
        "fielddata": [
                        {
                        "alarmtime": "2020-06-01 01:00:00.000",
                        "content": "测试线路 1.WKH-892 馈线保护测控装置.检修压板开入  投入",
                        "alarmlevel": 4
                        },
                        {
                        "alarmtime": "2020-06-01 03:00:00.000",
                        "content": "测试线路 1.WKH-893 馈线保护测控装置.检修压板开入  退出",
                        "alarmlevel": 4
                        }
                        ]
        }
}
```


返回数据失败示例：


```
{
"code": "08001",
"msg": "field 字段错误",
"data": null
}
```


C.3.5 审计服务接口

C.3.5.1 存储审计日志

1）接口定义

请求方法	POST
请求地址	audit-service/systemEvent/addAuditLog
协议应用	https

2）请求参数

消息头部包括类型、Token，应遵循 C.2 中消息头部定义。

请求体（Request body）为 JSON 格式对象，其数据的定义如下：

属性	名称	类型	必要性	说明
eventType	动作类型	String	必填	英文短语，记录动作类型，以对日志进行分类，具体由各服务定义，通用类型见下表
message	日志信息	String	必填	日志信息描述，描述具体进行了什么操作
detailMessage	详细日志信息	String	可选	详细日志信息，用于补充说明输入输出数据
auditObject	日志操作对象	String	可选	操作对象标识,比如设备 ID
serviceName	应用名称	String	必填	生成日志的应用名称
serviceAddress	应用地址	String	可选	生成日志的应用地址
userId	用户 ID	String	必填	用户 ID
userName	用户名	String	必填	用户名称
userRole	用户角色	String	可选	用户角色字符串
userIp	用户 IP	String	可选	浏览器 IP
userAgent	用户终端	String	可选	用户终端类型，与浏览器请求头 user-agent 保持一致
timeReported	日志生成时间	Integer	必填	记录日志生成时间的时间戳，精度到毫秒
isSucceed	操作是否成功	Boolean	必填	操作是否成功
severity	事件严重性	Integer	必填	事件严重性,正常操作为 0，不安全操作为 1，警告操作为 2，非法操作为 3，默认为 0

3)返回数据

返回数据遵循 C.2 中关于返回体的定义，业务返回码信息如下：

属性	名称	类型	说明
code	业务返回码	String	定义如下： 00000，业务请求成功 09001，日志存储失败
msg	业务码描述信息	String	详见 code 说明

返回成功示例如下：


```
{
"code": "00000",
"msg": "日志存储成功",
"data": null
}
```


返回失败示例如下：


```
{
"code": "09001",
"msg": "日志存储失败",
"data": null
}
```

C.3.5.2 日志事件通用类型定义

操作类型	说明
access	访问
create	创建
update	更新
delete	删除
reset	重置
disable	注销
active	激活
memo	记录
exec	执行
call	调用
install	安装
upgrade	升级
uninstall	卸载
overLimit	越限
overPower	越权
relate	关联
confirm	确认
download	下载
upload	上传
send	发送
receive	接收

附　录　D
（规范性）
数据通信网关机增加管理通道方案

应满足以下要求：

a）数据通信网关机提供与实时数据通道独立的管理通道。管理通道的端口号为 2403。

b）数据通信网关机应支持通过参数配置方式使能或禁用管理通道。

c）管理通道的链路工作方式应符合标准 IEC-104 规范要求，该通道应支持信息点表查询、全数据点表召唤、运行数据点表定制以及运行数据点表召唤。

d）实时数据通道应支持信息点表查询、全数据点表召唤、运行数据点表召唤以及模拟对点。实时通信链路不应支持运行数据点表定制。

e）可支持共用点表的多个实时通道复用管理通道完成点表定制。数据通信网关机的新运行数据点表生效后，应断开实时数据通道和管理通道，等待主站重新发起连接。

f）厂站的运行数据点表应基于主站下发的数据列表生成，应自动校验信息的一致性。

g）运行数据点表定制结束并通过自动校验后应立即启动生效过程，初始化时间应小于 60 秒。

h）运行数据点表生效过程中若发生任何错误，不应替换原有点表。

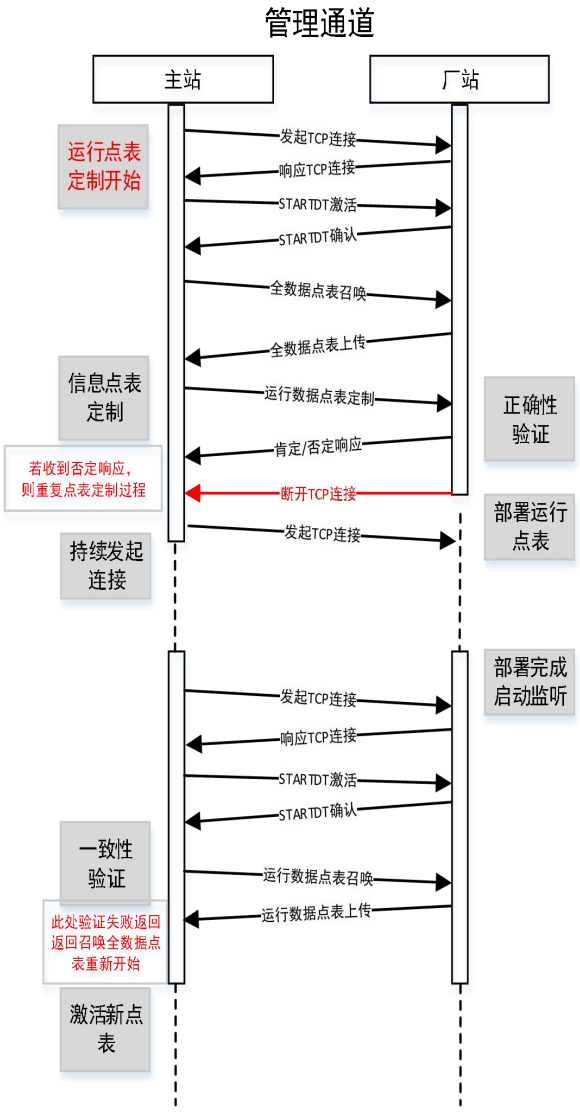


图 D.1 管理通道工作流程。

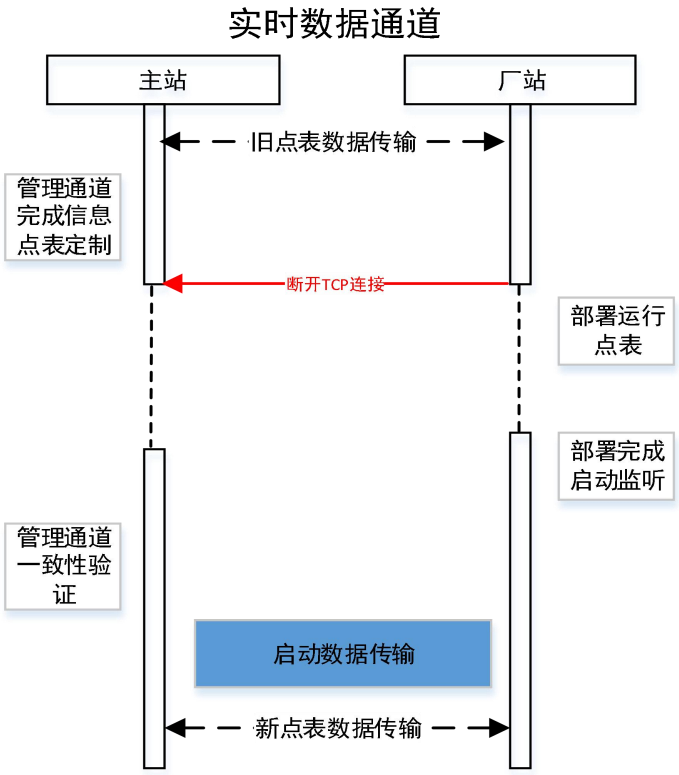

实时数据通道
主站
厂站
旧点表数据传输
管理通道
完成信息
点表定制
断开TCP连接
部署运行
点表
部署完成
启动监听
管理通道
一致性验
证
启动数据传输
新点表数据传输

图 D.2 实时通道的流程

附 录 E
（资料性）
原始报文转发

集控系统需要转发原始报文的直采链路划分为多个组，每个组共用一个 TCP 连接，降低原始报文转发复杂性。对原始报文进行扩展，添加链路标识、时标、版本信息。链路标识为通道号，时标为报文转发时的时间，精确到毫秒，版本信息为集控系统维护点表文件生成的时标，格式如下：

原始报文	链路标识（8 字节）	时标（8 字节）	版本（8 字节）

具体要求如下：

a）集控系统支持以转发原始报文的方式将信息发送给第三方系统；

b）采用分组的方式，将需要转发的多条直采链路，按规则（例如随机分、均分等）分配到各个组进行发送，每组建立一个 TCP 连接；

c）应支持多个服务器同时转发，各个服务器做为服务端监听多个端口（可配置），各服务器监听配置应尽量相同，由这些服务器共同保证全部链路已转发；

d）原始报文为集控系统直采链路接收子站的报文内容，不包含发送给子站的报文部分。